FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Cross-Browser Rendering using Headless Server-Side Browsers

Pedro Cardoso Lessa Silva

DISSERTATION

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Prof. Jorge Barbosa

July 11, 2016

Cross-Browser Rendering using Headless Server-Side Browsers

Pedro Cardoso Lessa Silva

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Doctor Rui Camacho
External Examiner: Doctor João Ferreira Sobral
Supervisor: Doctor Jorge Barbosa

July 11, 2016

Abstract

Nowadays many different browsers are used to navigate the Web, ranging from the most well known, such as Internet Explorer, Chrome, Safari or Firefox to less well-known browsers like Opera and SeaMonkey. These browsers are typically proprietary to the companies they belong to, and so, differ in their operations, of particular interest to this dissertation, the processing of HTML, CSS and JavaScript files which leads to the generation of different web pages, resulting in slightly different user experiences.

In order to homogenize the differences between browsers, the idea of cross-browser testing and rendering has been proposed and become a topic of research and work. Cross-browser rendering is the ability to generate a user interface also known as a web page, according to the browser specified. This idea has multiple uses, from providing developers with an easy method to validate their code under different execution environments to help detect security concerns of particular interest to this thesis, *Client-side Injection attacks*.

These types of attacks act at a DOM level in a user browser. Methods used to detect such attacks are based around code differentiation between an infected browser's representation of the website and the "clean" version provided by the web host. By leveraging cross-browser rendering, it becomes possible to minimize errors in malicious code detection algorithms by eliminating the rendering differences inherent to each particular browser.

This dissertation discusses the design for a system capable of performing cross-browser renders. Furthermore an initial foundation to this system is given, in the form of an HTML diffing tool used to identify and categorize the differences between renders of different browsers using this tool, and an analysis of current web automation tools both in performance and completeness.

Resumo

Hoje em dia são usados vários browsers para navegar pela Web, desde os mais conhecidos como Internet Explorer, Google Chrome, Safari ou Mozilla Firefox até browsers menos conhecidos como o Opera e SeaMonkey. Estes browsers são tipicamente pedaços de software proprietários pertencentes às empresas que os criaram, e como tal, variam no seu funcionamento, de interesse para esta tese, no processamento de ficheiros HTML, CSS e Javascript que leva à geração de paginas web diferentes resultando em experiências de utilizadores também diferentes.

De forma a homogeneizar as diferenças entre browsers, a ideia de testar e renderizar páginas de forma cross-browser foi proposta e têm sido alvo de investigação e desenvolvimento. Renderização cross-browser é a capacidade de gerar uma interface gráfica também conhecida como uma página web, de acordo com o browser em uso. Esta ideia tem múltiplas aplicações, desde fornecer desenvolvedores web um método para validar o seu trabalho em múltiplos ambientes de execução, até ajudar a detectar ataques informáticos, de particular interesse para esta dissertação são os ataques de injecção de código do lado do cliente.

Este tipo de ataques actua a nível do DOM no browser de um utilizador. Métodos actuais para detectar estes ataques baseiam-se em diferenciar o código de um site no browser e a sua versão inadulterada do servidor. Tomando uso de renderizações cross-browser torna-se possível minimizar o erro em algoritmos de detecção de código malicioso ao eliminar as diferenças de renderização inerentes a cada browser especifico.

Esta dissertação discute a arquitectura de um sistema capaz de gerar renderizações "cross-browser". Além disso é dada uma base inicial para este sistema sob a forma de, uma ferramenta que permite obter as diferenças entre renders de browsers a nível de HTML usado para identificar e categorizá-las assim como uma analise das actuais ferramentas de automação web tanto a nível de performance como de completude.

Acknowledgements

I would like to express my gratitude to my supervisor, Dr Jorge Barbosa, whose advice, knowledge and patience were crucial to this work. I thank him for his willingness to assist me, both with technical and non-technical fields.

I would also like to thank the head of the informatics department Dr João Pascoal Faria, whose guidance and mentorship during my graduate years proved extremely useful and whose friendship I will cherish. A very special thanks goes to the Informatics secretary, Maria Idalina da Silva, who was helped me more times than I care to admit. During both my time studying at Porto and abroad, she has been a steady support and helped in situations beyond her obligations.

I must also acknowledge Mozilla collaborates, Salvador de la Puente and Andrea Marchesini whose eagerness, to listen to a student talk about his crazy ideas and provide their very useful knowledge, incentivized me to continue this work when I started doubting.

To my close friend Luis Abreu, who somehow withstood my frustration venting and always had time for a friendly conversation. To my family, more than any another, without whom this thesis would not have been finished. To their unwavering confidence, support, encouragement and sacrifice to be there throughout my life.

To all these people, I dedicate my thesis as a thank you.

Pedro Cardoso Lessa Silva

"The Internet is the first thing that humanity has built that humanity doesn't understand, the largest experiment in anarchy that we have ever had."

Eric Schmidt

Contents

List of Figures

LIST OF FIGURES

Listings

List of Tables

Abbreviations

WWW	*World Wide Web*
HTML	*HyperText Markup Language*
CSS	*Cascading Style Sheets*
JS	*JavaScript*
DOM	*Document Object Model*
HTTP	*HyperText Transfer Protocol*
MitB	*Man in the Browser*
OS	*Operating System*
URL	*Uniform Resource Locator*
API	*Application Programming Interface*
XBI	*Cross-Browser Incompatibilities*

Chapter 1

Introduction

The use of the Internet has become a common activity for a lot of people. According to Internet World Stats, it is estimated that over three billion Internet users exist throughout the world [30]. Most of these users interact with the World Wide Web using pieces of software called browsers. As defined by the Oxford Dictionary, browsers are:

> "Computer programs with a graphical user interface for displaying HTML files, used to navigate the World Wide Web" [17]

Despite having the same objective, there are a number of browsers to choose from. According to GlobalStat's StatCounter[1] at the time of writing this thesis, 94.13% of tracked users use four browsers: Chrome (51.2%) developed by Google; Internet Explorer (17.11%) developed by Microsoft; Firefox (15.92%) developed by Mozilla and Safari (9.9%) developed by Apple, whilst the remaining 5.87% of users use other browsers such as Opera [31]. Each of these browsers are subject to updates and hence have multiple versions, Firefox alone contains over 40 major versions[22]. This means that a web page may be rendered differently based on browser vendor and version used(ranging from small layout details to functionality issues affecting the user experience). As an example, the HTML5 specification for the <video> tag came out in 2014, and it was not guaranteed to be supported by all browser vendors in the following months. To address issues like these, browsers are updated very frequently, which further increases the number of versions available.

[1] http://gs.statcounter.com/

These rendering differences between the different browsers are an issue to web developers, who without proper testing have no way of knowing how their web pages will be rendered across all browsers. For example: JavaScript events may not be fired, visual elements could be missing or misplaced in the screen, or even missing HTML tags when rendered in a different browser.

Rendering differences do not matter solely for developers who want their web pages to work correctly across-browsers. SaaS based companies who provide web services to improve the lives of these developers, also have an interest in handling browser discrepancies [9]. One such company is called JScrambler, they offer security and obfuscation solutions for web pages, and one of their challenges is how to handle all browser rendering variations. Particularly how to take it into account when protecting a client's users against cyber attacks. Consider how every single Internet user is liable to web-based cyber attacks, where their potentially sensitive information can be gathered by attackers. Given such a large pool of potential targets, it is no surprise that the Internet is the medium through which many attacks are perpetrated. This means that cyber security solutions such as what JScrambler offers must contemplate as many browsers as possible, in order to be effective against such violations.

In this paper we will talk about a very restricted subset of these attacks, namely *Client-Side Injection* cyber attacks also known as *Man-in-the-browser* (MitB) attacks. These attacks typically act on a web page's code in a user's browser. This means that in order to protect against these attacks, it is crucial to understand how the different browsers render a web page.

Due to a browser's complex architecture, there is no guarantee that the HTML content that is served from a server is shown correctly in the user's computer. The possibility of changing this content dynamically, accessing a web page's memory and modifying it, only makes analyzing HTML harder. According to the survey website W3Techs 92.7% of today's websites use some form of JavaScript [32]. Many technologies have been based around this language such as jQuery, Bootstrap, Modernizr, AngularJS, BackBone JS, React, among others. A list is available at w3techs[33]. With these technologies, websites become increasingly dynamic and adaptive to a user's interaction. Such dynamism creates many false positives when using naive malicious code detection algorithms, for instance, the same web page may change its code depending on a user's interaction. Hence any simple algorithm that detects code changes, between the browser's code and a clean version from the server, which does not contemplate this user interaction, would detect the code difference has potential malignant code which would not be the case.

In short, the security problem becomes then, how to detect and differentiate between harmless dynamically generated code and malicious code. A first step to solving this would be to have a browser process the HTML in a sandbox environment in order to get a baseline for comparison. Given the number of browsers and their potential differences, this solution would have to render web pages in accordance to the client's own browser. A problem which can be addressed by

attempting to figure out how to perform cross-browser rendering.

This cross-browser issue is the problem we attempted to solve. In order to do so, we propose a theoretical system and the initial groundwork which given an HTML document and a specified browser vendor will generate the associated DOM. In this way we hope to facilitate the interchangeability between browsers, allowing for a better basis of comparison for the detection of potential MitB code.

1.1 Context

This dissertation addresses the rendering problem described above in the context of the products developed by the company JScrambler. Their main focus is on JavaScript-based security and code obfuscation. One product provided by them is called *ZeroThreat*, which detects and handles MitB attacks. One of its core components consists of a server-side rendering service responsible for rendering the HTML documents received from a web host of the web page a particular client is accessing. The document object model (DOM) thus generated is later used as a baseline comparison against the client-side render done by a user's browser. This way, detecting potential MitB-related code can be flagged with more accuracy by removing from the analysis harmless browser-introduced code which could have resulted in false positives.

In order to be effective, the *ZeroThreat* solution must incorporate a rendering component capable of simulating the user's browser in a sandboxed and controlled environment. Given the nature of the Internet and variety of browsers available, this entails designing a solution which can simulate a large range of software and their versions. A simple approach to this solution would be to create a cluster of servers in which each machine would contain a unique browser and version. Upon a request to the service, a load balancer would decide where to send it in order to get the closest output possible to the user. However this pursuit has a few disadvantages, managing and scaling such a system would prove to be a very hard task, any update to a browser would have to be taken into consideration, forcing frequent downtime if not enough replicas exist. The cost of licensing the operating systems required to run some proprietary browsers such as Safari and Internet Explorer would also have to be taken into account. Given these restraints a second solution was proposed, a system based on headless browsers for improved performance, which would employ machine-learning techniques to train, adapt and minimize the differences between its own render and of the other traditional browsers.

However, despite the simplicity of the idea, we must take into account the variety of browsers and versions available to Internet users, so that we may obtain the best possible basis for comparison. This means, rendering a web page in a way that simulates that of the client's browser type

and version. Despite the fact that each browser receives the same HTML, CSS and JavaScript documents from a web host, assuming the latter does not have browser-specific code, the generated parsed document and functionality vary from browser to browser.

This is mainly due to the fact that most of the browser's code is proprietary to their vendor, meaning that each browser vendor implements the processing and rendering of HTML files in their own way. The list is quite big, ranging from different software vendors and their products, such as Internet Explorer, Google Chrome, Safari, Firefox or Opera.

1.2 Motivation

With web development being a key component of today's developer's lives and company products, testing their code is a vital and key aspect of their work. Having the ability to statically view how the code will convert to a browser's DOM, gives developers a tool to understand if their products will behave as intended.

The company JScrambler have identified certain problems whilst developing their products. The current rendering solution involves executing a Selenium [29] process in an Xvfb Ubuntu server (or X virtual frame-buffer server which implements the X11 display server protocol). Simply put, this means running an automated browser in a UI-less manner, which is similar to what headless browsers do. The first problem this solution has, is that it limits the rendering capabilities of the system to whatever browser is installed on the server. In addition, this method is limited to the support that Selenium has for browsers. By default this is merely Firefox, allowing only for renders supported by the Gecko engine [18]. With newer versions of Selenium Webdriver[2], this software is able to run tests in whatever browser and corresponding driver is installed in a machine. But because the system runs on a Linux system, it is incapable of running OS-specific browsers such as Internet Explorer or Safari. The second problem is performance, using Selenium when the UI component is unnecessary for the task at hand, makes this component cause needless overhead. For such a use case headless browsers could be considered as the interface component does not slow down the rendering process.

Although this project was proposed by JScrambler help to deal with MitB attack detection, we hope our solution will be able to go further. To this day, cross-browser web behavior has been difficult to test against, as a result of the rapid evolution of such software. With this thesis, we hope to develop a solution which can appropriately handle this dynamism in a way that is easier for developers to use.

[2]For more information http://www.seleniumhq.org/projects/webdriver/

Not only would this solution help with the *ZeroThreat* product previously mentioned above, but would also allow a web developer to see how their code would react under different execution environments. Designing such a solution as a service would also allow test driven development to be done easily, and in this way allow an easier method of testing cross-browser code and its behavior on multiple rendering engines by simply executing a request.

1.3 Objectives

It's our aim to outline the groundwork for a solution which can render web pages for multiple browsers, where differences between the multiple browsers' rendering engines are taken into account, in order to create a stable HTML-rendering system, which may be used as part of the *ZeroThreat* solution for code injection detection.

Due to the nature of the existing system, certain requirements for the solution have been defined. In the first place, the solution must be available as a service, which means it must be executed on a server and be transparent to a user. The solution must also have very approximate or equal rendering capabilities as a normal browser would, in order to perform as best as possible. As a third and final requirement, the solution to be developed must be able to render as many different browsers as possible and their range of versions.

1.4 Document organization

The rest of the paper is organized as follows. In chapter two we present an explanation of Man-in-the-Browser attacks. Moreover a formal characterization of the problem is given, with an explanation of all components and software involved in order to contextualize the reader, to how browsers work and what cross-browser technologies attempt to achieve. In section three we present the current state of cross-browser rendering and testing technologies and methodologies, its usability and how it can be used to solve our problem. In section four we present our proposal for performing cross-browser renders followed by an explanation of the empirical analysis be done in order to select the technologies in our solution. In chapter five we discuss HTML diffing algorithms as the detection tool for DOM-level inconsistencies and our work in the field. The results of both our analysis of the technologies to be used and our diffing tool are presented and discussed in chapter six. Finally, we draw conclusions and present improvements and future work in section 7.

Chapter 2

Technical Overview of Browsers and MitB Attacks

2.1 Man-in-the-Browser attacks

Man-in-the-browser attacks (MitB in short), are a specialized subset of Man-in-the-Middle cyber attacks. These malicious pieces of software act by insert malware into a victim's computer browser. These types of attacks usually target banking services due to their ability to be able to

> "steal authentication data and altering legitimate user transactions to benefit the attackers" [16].

Once having inserted themselves there, these pieces of software are able to perform a variety of actions. They may listen to all traffic between a user and a server, by acting as a proxy in the communication between these and stealing sensitive information such as passwords. They are also capable of creating and modifying HTTP requests which the computer's user had no intention of doing and without his or her knowledge.

This type of malware will act using one or more methods to achieve its purpose. From API hooking, inserting JavaScript code snippets into a given web page's source code, to DOM exploits via DLLs or Extensions. Each of these attack vectors allows the offenders access to private, and seemingly secure information about the user's interaction, with that particular web page. Because these attacks originate from a browser, their detection is a hard to accomplish objective.

That does not mean however that their detection is impossible. Whilst from a user point-of-view, an infected page may appear normal, it's internal content is most likely altered. If a MitB attack operates through a page's DOM or through API hooking, it means that the code that defines a web page in the browser is different than what it should be. This means that if we can generate a sandboxed version of the same page, where the browser is protected from malware, then we obtain a clean version. With both of these versions, it would then be possible to compare their DOM and identify the discrepancies between them as altered and potentially malicious code. To create such a detection method, the first element to address is how to create such a sandboxed version of the web page. In order to increase the effectiveness of this system, the "rendering" component has to simulate as closely as possible, a user's browser without the malware infection. This introduces the need for cross-browser rendering which will be extensively talked about in this paper.

2.2 Browser architecture

In order to understand how cross-browser rendering works, we must first establish how a browser works.

When a user enters an URL or clicks a link in a website, a HyperText Transfer Protocol (HTTP) request is done to the remote web server of that specific URL, in order to retrieve content to be displayed. Usually, the server then returns a HyperText Markup Language (HTML) document which includes references to any other dependencies it may have, such as CSS, JavaScript or other types of files. These files provide additional layout and style information and client-side computing abilities respectively, used to give the user a more immersive experience. There also exist other components to a browser, such as plugins, bookmarks, cookie and password management which serve functionalities not directly connected to a specific website but which also assist the user in using the software.

After an HTML document has been received by the browser, it shall be parsed, interpreted, rendered and finally displayed on the screen. In order to do this a browser must have some components which allow these operations, figure 2.1 shows a high-level conceptual architecture of these components.

Browsers nowadays contain more components than the ones shown in the figure, however, for the purpose of this thesis, we shall briefly describe what a few of these subsystems do, namely: the Browser Engine, Rendering Engine, Data Persistence Store and JavaScript Interpreter are of interest to this dissertation. The Browser Engine provides a high-level API for the communication and manipulation of the Rendering Engine. This component in return is responsible for parsing HTML documents using the API provided by the HTML parser, creating internal data structures of

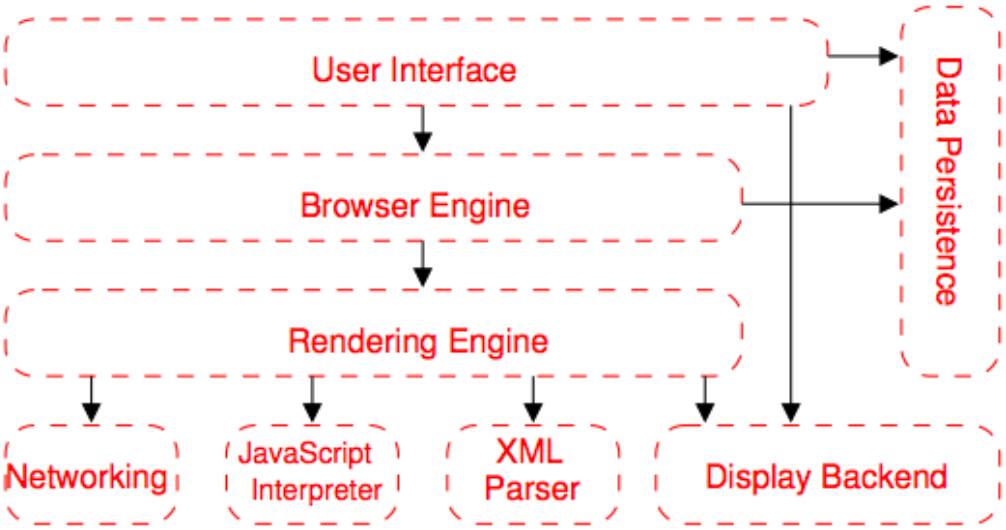

Figure 2.1: Browser Architecture, drawn from [19]

this information which include how the document shall be represented graphically (including CSS layout information). The Data Persistence Store saves a variety of data associated with the user's interaction with the browser including cookies, browsing information, history records, bookmarks and cache. Finally, the JavaScript Interpreter is charged with validating, compiling and executing any JavaScript code which may be linked to the HTML document received, including anything from dynamic layout changes to event handlers.

When an HTML file is received through the network component of the browser, which handles all low-level network connections, it is sent to the browser engine. During this process, the document is analyzed by the XML parser typically in chunks. As the document is parsed, it is the engine's job to deal with any links it may have. These come in the form of HTML script and stylesheet link tags, which may have an influence in the document's appearance after rendering.

This workflow is done recursively, for any link to an external dependency the document may have. If such a dependency is detected, the browser engine must: (1) request the file via the networking module, (2) await the response, (3) parse the new document, (4) process it, making the necessary changes to the user interface as necessary and, finally, (5) proceed to process the original file with the dependency. 2.2 shows an exemplary HTML document with a JavaScript file as a dependency, if executed, an alert notification will appear and the second paragraph text will only after dismissing it.

In the example shown, the dependency and processing blockage of the HTML file is demonstrated with the use of a JavaScript file. As mentioned before, these files are used for client-side

```
<html>
<head>
 <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.0/jquery.min.js"></script>
 <link rel="stylesheet" type="text/css" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css">
</head>
<body>
 <p>This is an example code</p>
 <script src = "popup.js"></script>
 <p>Until the alert popup has been removed, this text will not appear</p>

</body>
</html>
```

Figure 2.2: Basic HTML example, with dependency processing.

```
function load() {
 alert("load event detected!");
}
load();
```

Figure 2.3: Depending popup.js file.

computation, anything from enriching the interface a user sees through animations, changing the CSS properties of specific DOM elements to input validation. In order to do this, a JavaScript Interpreter will parse, validate and execute any JS code that is detected by the browser engine. This interpreter uses just-in-time compilation to convert the JavaScript code into the appropriate low-level presentation which is executed, the executable code can then be used to alter any aspect of the correctly displayed in the user interface. One reason why MitB attacks are effective is because JavaScript code has the ability to access unencrypted, text-based information that is stored in the DOM as well as cookies pertaining to the web page in question that may be stored in the browser.

However, not all documents are interpreted and rendered in the exact same fashion and with consistent outcomes. This fact has lead to research in how to detect such situations as shown in figure 2.4. As a general rule both academia and industry have focused on how to detect and test for such situations in order to minimize these inconsistencies. This issue arises from the browser's components briefly mentioned above.

Because most browsers and their underlying components are closed-sourced and unavailable for analysis, it means their output varies depending on the vendor who implemented it, typically for competitiveness. Despite existing specifications made by the W3C[13] organization on how HTML should be interpreted, there still exists some discrepancies in some browser's actual implementation of how to correctly process web pages [20]. Of particular interest to this thesis are the rendering engines used by browsers and the discrepancies between them.

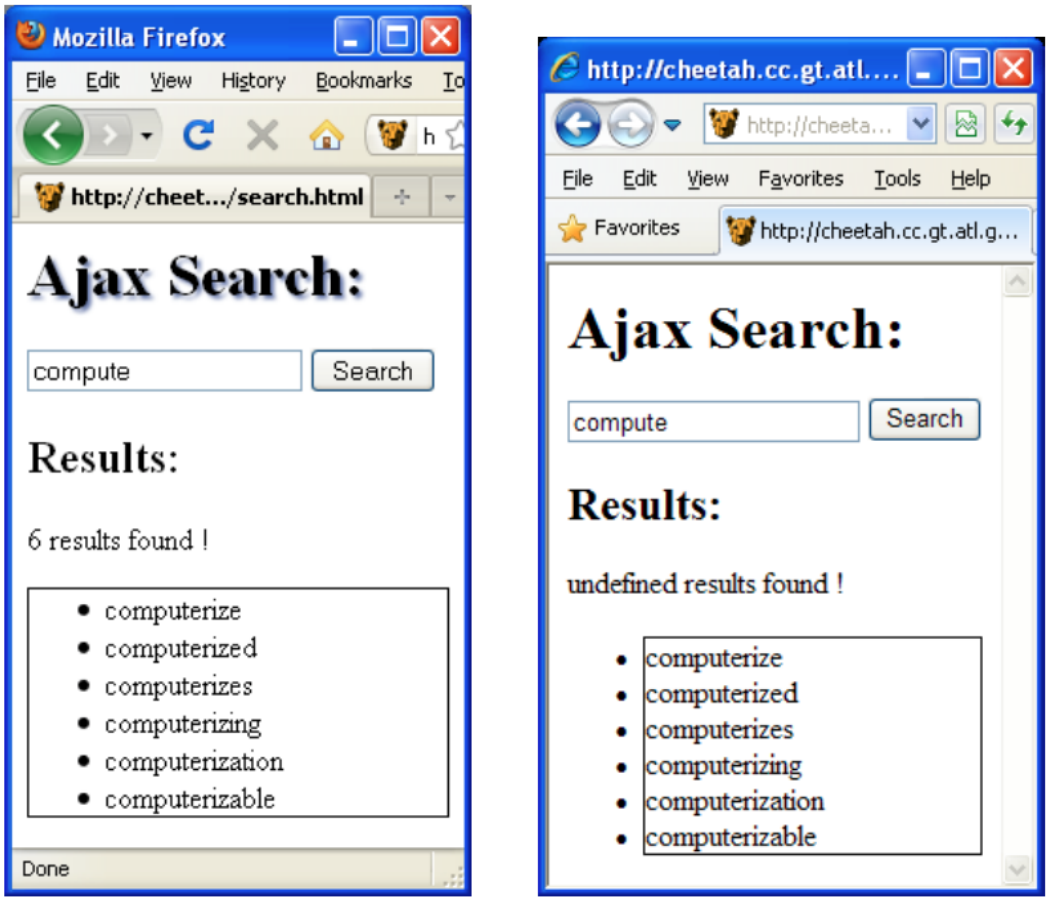

Figure 2.4: Rendering differences in Mozilla 3.5 and Internet Explorer 8, extracted from [11]

Though there is little information about the implementation of the most common browser's rendering engines, they often are based on open-sourced projects. For example, Google Chrome and Opera 15.0 have engines based on Blink which itself is a fork of Webkit, Mozilla Firefox's engine is Gecko and Internet Explorer uses Trident (unfortunately not open-source). The code for the first two can be found online via [1] [4] and [18].

2.3 Headless browsers

Headless browsers are, essentially, web browsers without a graphical user interface. This means no user interaction is done to access, interact or navigate a web page. Instead, headless browsers are used legitimately to provide the content of web pages to other programs. For instance, Google has said that using headless browsers is a way to enable its search engine to cope with *Asynchronous JavaScript and XML* (AJAX) code [8]. Headless browsers are also useful for smaller tasks, they are

Table 2.1: Characterization of common potential headless technologies

	Selenium	**PhantomJS**	**SlimerJS**	**Awesomium**	**HtmlUnit**	**TriffleJS**
Language used	Multiple	JavaScript	JavaScript	C++	Java	JavaScript
Headless mode natively supported?	No	Yes	No	No	Yes	Yes
Render engine used	Uses the browser installed in the machine	QtWebKit [35]	Gecko	WebKit	Rhino	Trident

used by regular developers who intend to test their code and simulate user navigation & interaction for example.

In the same way, browsers vary amongst themselves as mentioned in the previous section, so too do headless browsers. There are a few browser-like technologies which could be considered headless because they do not have the performance cost of running a full browser. Such examples are PhantomJS, HtmlUnit or TriffleJS, each of these web browsers automation tools may be run without a UI in order to improve performance by reducing the processing overhead of constantly refreshing a web page after an action or event has been triggered.

From the research made, we have come to the conclusion that many web automation tools are not natively headless, however, the functionality can be simulated by running the software in servers without a GUI. A typical example of this is using Linux production server running the processes with Xvfb, note that this is only possible in Linux or Mac OSX operating systems. In the table 2.1, a brief detail of the principal characteristics of web automation tools are given.

When searching the web for web automation tools, we noticed seen that Selenium and PhantomJS were two of the most popular tools. A quick search on Google for web automation returns Selenium as the first link[1]. A similar story occurs for PhantomJS if we search for headless automation[2]. Further review of each of the tool's web pages and related reviews make their popularity and success apparent. No other search for web-related automation tools achieved such results.

Whilst both do very similar things, their main contrast is the rendering engine used. Selenium works as a tool to automate the use of a browser previously installed in the machine, using

[1] https://www.google.pt/search?q=web+automation+tool&hl=en&authuser=0

[2] https://www.google.pt/search?q=web+automation+tool&hl=en&authuser=0#safe=strict&hl=en&authuser=0&q=headless+web+automation+tool

the browser engine's API in an RPC-like manner. In order to use Selenium for web automation, a driver and corresponding browser must be installed. These drivers are sets of mechanisms used to control the browser varying from browser to browser, e.g: Firefox control is done via a plugin whereas controlling Internet Explorer is done via a standalone server which implements WebDriver's wire protocol [28]. Selenium has one major advantage in testing, because of it's dependency on the browser installed in the testing machine, it is able to correctly render a web page for that browser, e.g: If Firefox is installed and used, then the output of the execution can be expected to be exactly what that version of Firefox would give.

PhantomJS, on the other hand, is an open-source natively headless port of the WebKit engine with a JavaScript API [23]. This means that only a subset of the main components used in browsers are executed in PhantomJS. Because it is natively headless, PhantomJS has both an advantage and disadvantage over tools like Selenium. The cost of booting up a browser instance each of time a script is executed does not exist in Phantom, the script executed directly on the WebKit engine leading to reduced overhead. On the other hand, because PhantomJS uses its own rendering engine, it is incapable of outputting directly the DOM corresponding to a browser which uses a different rendering engine, e.g. Accessing the web page `www.google.com` would most likely output the DOM corresponding to Google Chrome (because PhantomJS derives its engines from WebKit) but could be different from the output of Firefox or Internet Explorer, whose engine's are Gecko and Trident respectively.

There are other automation tools, many of these being derivations of PhantomJS which merely extend some functionality or change the execution environment. CasperJS, for example, is a wrapper JavaScript utility tool built to work on top of PhantomJS or SlimerJS for easier use of both of these tools. Ghostbuster is an alternative to CasperJS. SpookyJS presents itself as a NodeJS driver in order to run CasperJS processes from within Node source code. There are many more, a simple search for headless browsers show many interesting results, there is also a compiled list[3] of some of these headless browsers available.

[3] `https://github.com/dhamaniasad/HeadlessBrowsers`

Chapter 3

Related work

Now that we have a clearer view of the problem and context at hand, we can go into detail over what has been done in the area of cross-browser usage. Because the problem at hand is quite specific, there has been some difficulty to find research on this particular field. However the existing investigation on cross-browser testing provides a starting point to analyze what has been achieved thus far. In this section, we look into the current solution the company JScrambler has been using, and the research done in cross-browser testing. This information will then be used as a starting point to explain the design of our solution.

3.1 Current solution

The current solution employed by the company JScrambler as briefly mentioned in chapter one involves using Selenium in a Linux server executed in X virtual framebuffer (Xvfb) mode. This means running a terminal based system in which UI operations are done over a block of memory and not sent to any display peripheral. Using this approach is akin to running a normal browser on a computer and downloading the DOM generated. By running Selenium in this manner the system effectively executes all steps of the browser workflow despite having no "real" user interface. Thus the system suffers from both the overhead of launching a new browser instance for each execution, and performance problems due to having to run graphical primitives to draw a UI that is not visible.

These issues do not allow the component to perform adequately for a real-time DOM generation for posterior comparison. By changing the system to use headless technology we hope to minimize this overhead problem. Another particularity is the nature of the server, Internet Explorer is not a render option available to detect Trident-specific code in Linux systems.

3.2 Cross-browser testing tools

The cross-browser area of investigation has been looked almost exclusively from a detection-of-problem point of view. By this, we mean that, researchers and investigators have focused on the existence of multiple browsers, and their implementations as the origin for cross-browser incompatibilities (XBIs) -

> "situations in which the same web application can behave differently when run on different browsers. In some cases, XBIs consist of tolerable cosmetic differences. In other cases, however, they may completely prevent users from accessing part of a web application's functionality." [10]

This research has led to the creation of tools, which discover and report discrepancies in execution between browsers, of which we will briefly talk about three: Webdiff, CrossCheck and XPERT

3.2.1 Webdiff

Webdiff was the first of the three tools we looked at. Published in a paper in 2010, it is a tool designed to detect cross-browser visual problems. Webdiff was a significant improvement to the tooling available at the time which required manual inspection and effort from a user's point, by using two techniques in their algorithm.

The first, DOM element matching, where two DOM nodes from different browsers would be matched to each other using the node's attributes. This technique would allow the tool to make a structural analysis of each DOM, in order to detect if a certain node was compatible in both browsers. This would report an issue if a browser did not support functionality, special tags or properties.

The second technique was to visually locate the element-pairs generated by the first technique and perform a visual analysis from screenshots of the two browsers being compared. Using a pixel based differentiation algorithm, this tool can see if any element was correctly rendered or not in a specific browser. Furthermore, it can detect variable elements such as ads by accessing the web page multiple times and detecting regular visual changes. In this way, it is able to ignore such sections and not report these differences to the user.

With these two techniques, Webdiff was able to report a significant number of XBIs. From the experimentation and evaluation published in the paper [11] the tool was able to reportedly detect 121 XBIs in a set of nine different web applications, ranging from university web pages to e-commerce, out of which twenty-one were false positives (17% of false positives).

3.2.2 CrossCheck

The second tool CrossCheck [10] published in 2012, presented an improved version of Webdiff. This tool improved upon the shortcomings of Webdiff while adding functional XBI detection from a tool called CrossT [21]. CrossT was first described in a paper about how to use dynamic web crawling and navigation models, to detect which functional use cases were broken in a browser.

CrossCheck does not only adapt the XBI detection techniques from previous papers, it uses machine learning and a new metric, which detects visual discrepancies in order to improve the visual analysis. By performing multiple tests on the output of the browsers, CrossCheck was able to go a step further in regards to the others tools. Not only by being more accurate in the XBIs it detects, but also by being able to inform the user of the type of inconsistencies detected. In the paper, the authors compare all three tools for the same set of web pages, with CrossCheck outperforming both its predecessors, due to the improved visual metrics and machine learning-based classifier used in the comparisons.

3.2.3 XPERT

The final tool, XPERT, [26] is the third and final XBI detection tool we will discuss. In the same way, that CrossCheck evolved from Webdiff and CrossT, XPERT is the natural successor to CrossCheck. XPERT was published in 2014, this tool allows the detection of three types of XBIs, structural, content and behavior.

Structural XBIs are the cause for layout or incorrect placement of elements in web pages (first detected by Webdiff), these are the most common types of cross-browser inconsistencies. These XBIs are detected using alignment graphs which allow the tool to detect the relative arrangement of elements on a screen. The algorithm used to detect this is described in [25].

Content XBIs relate to the value which DOM elements contain when generated in a specific browser, as specified in the paper this can be further divided into two groups: visual and text based. Whilst being simple to detect in theory, they could sometimes be overlooked in the previous tools because of frame scrolling. This means that the browser screenshots used for the visual

comparison did not contain these elements. To deal with textual XBIs, XPERT extracts all text values from a DOM, associating it with their element reference in an internal representation and then matched to the other browser's DOM values. Visual XBIs are detected using CrossCheck's approach but only applied at a DOM-level in order to be as effective and performant as possible.

The final type of XBI, behavioral, is defined as variations in the behavior of widgets in a web page. This can be elements which do not perform the same actions in a different browser, this could be a button which does not perform the correct or any action or links which become broken. Such XBIs are detected through a combination of web crawling, where the tool will attempt to execute all elements in a web page and from the navigation models generated from both browsers, compare them using graph isomorphism and report where a divergence was detected.

From the paper, the authors show that XPERT accomplishes its task. It is able to effectively detect all three types of XBIs. Furthermore, when compared to CrossCheck, XPERT was more accurate in detecting visual and functional XBIs:

> "A deeper investigation of the results [6] revealed that X-PERT's precision and recall are 76% and 95%, respectively, against 18% and 83% for the state-of-the-art tool CrossCheck" [26]

3.2.4 Usefulness

During the initial research for this thesis, it appeared that the work done in the field of cross-browser inconsistency detection would be of great help to the problem at hand. Having a system capable of reporting differences between browsers would be extremely useful when designing a solution, which would then learn from these inconsistencies, create transformation rules, which when applied would minimize such differences. However, such a use was not possible for a few reasons.

The research and investigation which has lead to the development of the tools mentioned above, arose from a need to understand and test how websites behaved in different browsers. The focus and purpose of these tools were on how to detect cross-browser inconsistencies in the multiple forms these appeared in: visual, syntactical and behavioral incongruences. These inconsistencies were detected by comparing the outputs of different browsers and if divergences were found it would be up to the user of the tool to decide which is the correct version. In a self-taught system such a decision must be taken automatically and from the data generated by the tool, no assumptions are made about which is the correct output.

Of the three tools, only one was available for analysis and use, namely the XPERT project[1]. At first, this was encouraging as it appeared to be the most efficient of the three tools and with the greatest effectiveness of detecting XBIs. When looking deeper on how the reporting of XBIs was done, it became apparent that the data outputted from the program was unyielding and non-trivial to feed into an automated system. An example of the output from the XPERT tool such as state transition graphs and HTML based reports can be seen figures 3.1 and 3.2 respectively.

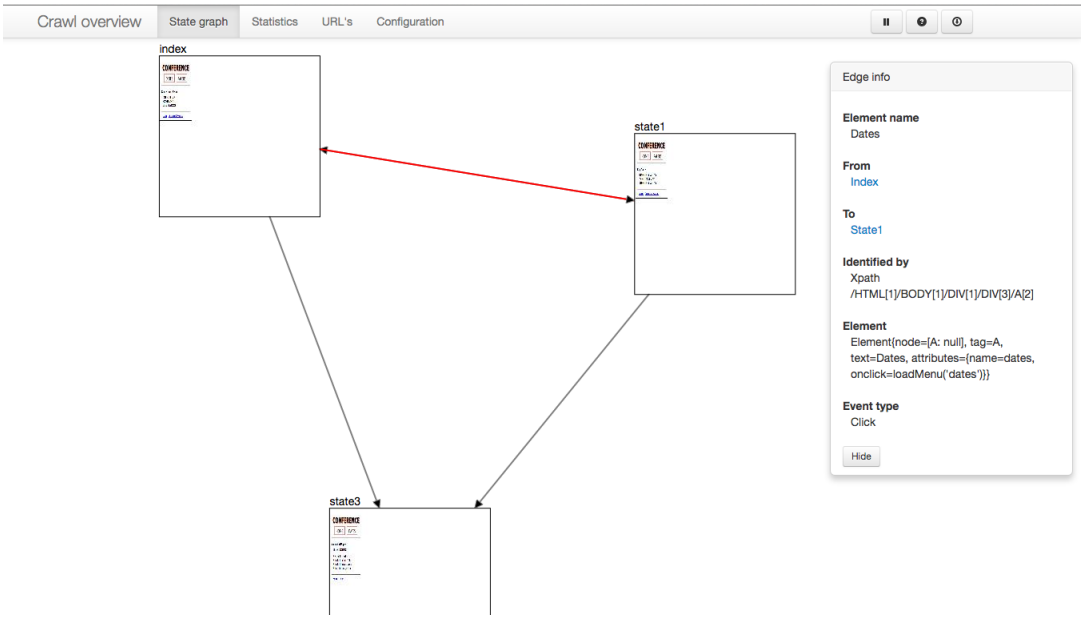

Figure 3.1: XPERT state graph output example

As you can see, the big emphasis of these reports was on the transitions between the multiple states a given website can have. Whilst such information is useful from a web developer's point-of-view, to understand what pieces of code are not functioning as expected, it does little to tell the user exactly why that occurred. In addition to this, the XPERT tool's methodology, based on state transitions, means that the granularity of how browsers' basic components and elements work are not explored. For instance, with the XPERT tool, differences at an HTML syntax level are not detected. The tool only reports that for a given state, a component and/or element is missing. Apart from the data that is provided by the tool, after taking a look at the source code, it became apparent that the tool was built with a heavy reliance on Selenium to communicate and dictate the actions of the browsers being used. This created an extra obstacle to overcome, namely how to expand the tool to work with other technologies such as headless browsers, of which PhantomJS is a very relevant one.

In conclusion, the XPERT tool reports errors but gives little useful information as to the origin of the problem. Furthermore, the time and effort needed to extend the tool to include additional

[1] https://github.com/gatech/xpert

Figure 3.2: XPERT state summary example

technologies such as PhantomJS outweighed its advantages in this work.

Chapter 4

Adaptive Browser Rendering

4.1 Overview

To deal with the cross-browser rendering inconsistencies, first mentioned at the start of this document, we propose the idea of an adaptive browser. Simply put this system is a browser with the ability to simulate the processing of files similar to what is done by browsers today. From the related work chapter, we have seen that there are inconsistencies between browsers, that these have an impact both in web development and UX, and that some research has been done but that it is focused on the detection of such problems and not how to deal with them. In this chapter, we shall explain the design of this system and how to converge differences between browsers.

4.2 Design

In order to design and create a system capable of simulating the rendering of a DOM, from a given HTML document and its dependencies for a specific browser, we must first have an initial basis with which to work. Because browsers are extremely complex pieces of software we chose to select a browser technology which will serve as our initial and basic starting point, instead of implementing every single component, as this would be redundant and error-prone work. The objective is to compare the selected base browser's output with other browsers in three fields: HTML CSS and JavaScript. The hypothesis of the thesis is that simple differences cross-browser detected in these fields compounded on one another are the basis for a cascade of differences, detected later on by many tools and reports (CrossT, CrossCheck, Xpert). To make the system as

complete as possible, the system being built must be done in such a way, as to automatically and without supervision learn, from parsing, interpreting and executing each of these file types.

After having selected the basis of our tool, we must create the necessary tooling for the system to understand what it's performance is, both qualitative and quantitatively, relative to other browsers. These tools should evaluate the output from our own system and compare it to the output of other browsers, and if differences are detected and are consistent throughout multiple tests. With these reports, the system should then attempt to induce transformation rules from our underlying browser to the browser being tested. The design diagram 4.1 illustrates the components necessary for this work and how they interact between themselves.

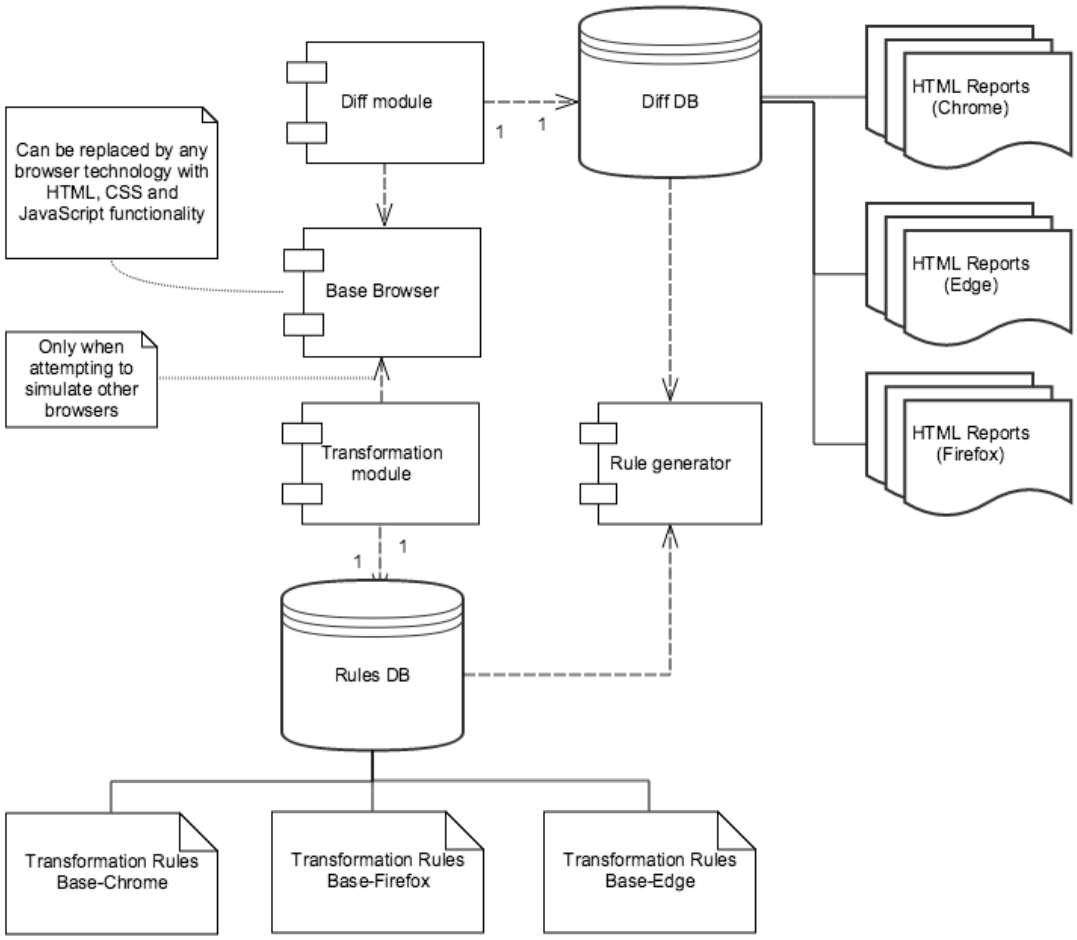

Figure 4.1: Adaptive Browser Component Model

4.2.1 Browser options for adaptive system

For the selection of the browser technology the following criteria were taken into account:

1. Must have a self-contained HTML renderer and JavaScript interpreter so that basic code manipulation can be applied (such as running JS scripts to test if certain features are available).

2. Reliability and stability of technology to ensure that the foundation of the system is adequate for real use and not a critical point of failure.

3. Ease of use for automation and development.

4. Active support and community, to ensure the project is kept up-to-date, with the rapid evolution of the Web and to rapidly fix any bugs that may emerge in this environment.

Initially for this work, the following were looked at: Selenium [29] which is the name for a suite of applications designed for web browser automation. PhantomJS [23] a headless WebKit-based tool. SlimerJS a scriptable browser based on Firefox's rendering engine Gecko [6]. Though the latter is not headless by default it can run with UI in Linux and Mac OSX. Lastly Awesomium an HTML UI Engine written in C++/.Net will also be considered.

In the following sections, we shall talk about our findings of these four options based on the criteria given.

4.2.1.1 HTML renderer and JavaScript interpreter

While this criterion might at first appear strange to the reader, during research it became apparent that whilst the presence of an embedded HTML renderer and JavaScript interpreter are inherently available to any browser, the same does not happen in "browser-like" options. Our objective is to create an adaptive browser, and in order to do so we must be able to manipulate its output after the initial web page processing has concluded. This is so we can validate its output by executing specific tests. For instance, with the HTML5 specification certain features were added, like new dynamic HTML elements and support for different audio encodings. The methodology used to test support for these features came in the form of JavaScript files. The idea surged from a popular JavaScript library Modernizr [5], which is essentially a series of JavaScript tests intended to verify if certain features are available in any given browser. This code works through "shims", which are backwards-compatible pieces of code that work across multiple browsers, the tests used can be seen here [2].

In this criteria, the library Awesomium was rejected because of the C++ bindings needed to manipulate JavaScript code, which in turn is required to retrieve information about the rendered page (see figure 4.2).

Because Awesomium uses WebKit as the underlying HTML rendering engine, no extra information could be extracted from its use that couldn't be done with PhantomJS (which also uses WebKit) or Chrome via Selenium (for this component in the system). Ergo it was decided that the extra work needed to add Awesomium to the testing framework was not worthwhile.

Create Global JavaScript Object

Let's create a special JavaScript object that will stay alive throughout the entire lifetime of our WebView. This will allow us to expose certain methods and properties that our pages can use via JavaScript.

```
// Inherited from Application::Listener
  virtual void OnShutdown() {
  }

  void BindMethods(WebView* web_view) {
    // Create a global js object named 'app'
    JSValue result = web_view->CreateGlobalJavascriptObject(WSLit("app"));
  }
```

And then add the following code underneath your LoadURL code:

```
// Inherited from Application::Listener
  virtual void OnLoaded() {
    view_ = View::Create(500, 300);

    WebView* web_view = view_->web_view();

    BindMethods(web_view);

    WebURL url(WSLit("file:///C:/Users/awesomium/Documents/app.html"));
    web_view->LoadURL(url);
  }
```

Figure 4.2: Awesomium JS binding for global objects [7]

4.2.1.2 Performance, Usability, Reliability and Stability

Having a stable environment is critical to the system we are attempting to build. Simply because of the sheer number of potential points-of-failure. If the base case does not perform as expected it can

invalidate the work done by the other components of the system, and output incorrect information without having any mechanism able to correct it.

In this case, SlimerJS did not function as expected. The current API available to work with is inadequate for intensive usage, as no garbage collection can be explicitly done. Making executing a tool with this library for multiple files/URLs overload the system's resources. Requests to change this have been filed, but due to a lack of contributions and development of this library, it is unlikely that such change will occur during the time of this thesis. Another reason that lead to dropping SlimerJS was the lack of benefits performance-wise, for a headless solution. Running this library will create a window in which to render the requested page. The advantage of this according to the developer, was that any update to the firefox browser will be present in SlimerJS, allowing for security and performance upgrades to come, however for the current scenario, it makes it no different than using selenium with Firefox.

During development of the testing framework and subsequent execution of the tests, both PhantomJS and Selenium functioned as intended, in that their APIs were very similar and allowed almost the same code to be used. They were also reliable and easily manipulated. Further testing was done in regards to reliability and performance, for more information please see the Results chapter.

4.2.1.3 Support and community

In terms of active support and community, both Selenium and PhantomJS options provide interesting scenarios. Because Selenium is a wrapper to actual full featured browsers, support for new technologies bug fixes are done by the developers of this software, namely Mozilla, Apple, Microsoft and Google to name a few. An update to the browser installed on the system would imply that the adaptive system on which it relies would also automatically update with little extra work. PhantomJS on the other hand, to become up-to-date with newer features, takes more time and in some cases, the developers decide not to implement them at all (see [24]).

4.2.1.4 Additional observations

After the experiences mentioned above with simple testing and usage, we were left with PhantomJS and Selenium. For Selenium we chose to test two browsers Google Chrome and Firefox, the reason being that they are the two most used browsers at the time of writing this thesis, and also very importantly, is that both options work across operating systems.

To note that, here neither Internet Explorer nor Safari are included in the Selenium option. This is because both browsers and corresponding webdrivers [3], require a Windows and Mac OS operating system respectively, making it inadequate for server deployment on Linux systems. Furthermore, Internet Explorer does not allow the possibility of opening local files through the file:// schema used (see Results chapter) to process our test suite. This, however, does not exclude either browser from being used to "show" the system it's differences, merely from using them as the "base" browser.

4.2.2 Rule detection and transformation

The first step in designing the rule and transformation modules is to understand where the starting point for divergencies between browsers occur. As mentioned briefly in chapter two, each browser's rendering process starts with the user's input of an URL and the network library requesting said file. After the file has been received it is processed by an HTML parser.

It is at this stage that differences may begin to occur. HTML parsing is theoretically done by following the W3C standard. However the standard is continuously evolving and regularly revised (see [15]), and so, it is not unreasonable to assume a possible mismatch between the HTML standard today and a browser parser's compliance to the standard. During this parsing, information is passed to the DOM, where it is converted into actual in-memory node/elements, which is what the browser uses for posterior processing (usually JavaScript). This conversion from HTML to DOM objects is done in a one to one relationship, meaning that there is a direct match between the two representations. Hence the first point of entry for differences between browsers is in the HTML parsing and subsequent in-memory representation.

In order to detect these variations at a DOM level, we began by creating unitary tests, that focus on testing each of the building blocks of HTML at a granular level, specifically at the tag level. In this way, we shall attempt to reason later on about the output of each browser technology for each tag type. These tests were based on those defined by w3schools[1], for each element, the corresponding example code for that tag was used. In some cases the code was adapted so as to remove unnecessary tags, in most, however, the code was used as is. An argument can be made to put into questioning the impartiality of these pieces as unitary tests, and their integrity in such role, especially when considering that some do not merely contain just the tag to be tested. However the reader must also bear in mind that some tags have an inherent dependency on others, or require additional code to adequately show the functioning of the tag, for example, the <tr> or table row tag does not make sense without being wrapped in a <table> tag, nor does it comply with the standard. By using the code samples from w3schools (a certified web teaching organization[2]) as is, we can be sure that the samples reliably demonstrate the use of that specific element.

If any of these tests generate relevant (for more details see the Results chapter) differences between the base browser and the browser being tested, we shall look to see if it is possible to induce transformation rules for each idiosyncrasy found. Following this methodology the system should then, in theory, be able to provide to cross-browser compatibility at a DOM level in the following manner:

[1] http://www.w3schools.com/tags/
[2] http://www.w3schools.com/cert/default.asp

$$P_{xi} = T_x(P_{xs}) \tag{4.1}$$

Where:

- x: is an HTML Tag defined by the HTML5 Standard [14].
- i: is the browser being used to teach the system.
- s: is the base browser on which the adaptive browser system is based.
- P_{xi}: is the processed DOM code of tag x by the browser i.
- T_x: is the transformation rule for a given tag x.

The detection and validation of these transformations HTML files will be the focus of the work presented ahead for two reasons. Firstly, it is the basis upon which CSS and JavaScript work, hence without adequate knowledge of how HTML is processed no adequate comparison can be made for the other two. Consider a case where for a given element such as <datalists> (which are not supported in IE9 or lower), which when processed by firefox creates additional elements in the DOM. This has implications on the JavaScript interpretation of that element for the firefox browser which are not found in Chrome or PhantomJS. On the other hand, HTML is by nature static (when processed in standalone fashion without scripts), and consequently the easiest of the three file types to handle in order to test the validity of unitary comparison for cross-browser interoperability.

In the next chapter, we shall look into a tree-based diffing algorithm, in order to compare the output between our render and that of other browsers. With such "diff" results we will determine their usefulness towards creating transformation rules that will enable cross-browser compatibility in HTML files.

Note that this approach does not cover other potential cross-incompatibility points of failure, such as data storage or UI primitives. These are not part of the scope of this paper nor accomplishable without more in-depth research into the side effects of browser components.

Chapter 5

Tree-based diffing

So far, in order to create the system we designed, we have an automated rendering framework which given an HTML file and a specified browser will process its content and any external files it to refers to. Having done so, it is necessary to evaluate the output of this render and compare it to with our own system's default option. These type of file comparisons are typically done through "diffing" tools, which use special algorithms to calculate the Levenshtein distance [27] between two strings. These utilitarian pieces of software are comparison tools that calculate and output the differences between two files, the most common one being UNIX diff for raw text.

Because HTML's structure is intrinsically a tree data structure, it means raw textual "diffs" do not report adequate results. For example, consider variations in HTML files such as white spaces, comments (in the form of <!– –>) and different order of attributes in a given element that may be outputted, by text-based tools. These are in fact false positives and do not have an impact on the interpretation of the file. In a similar way, meaningful differences are frequently not detected adequately. For instance, see below how changes between two HTML files (presented in Listings 5.1, 5.2), at a content level of nodes are outputted 5.3, and how easily or not it could be made useful for an automated system.

```html
<html>
  <head>
    <title>This is a Title</title>
  </head>
  <body>
    <p>Hello world!</p>
  </body>
</html>
```

Listing 5.1: HTML code sample 1

```html
<html>
  <head>
    <title>Another Title</title>
  </head>
  <body>
    <p> Bye World</p>
  </body>
</html>
```

Listing 5.2: HTML code sample 2

```
< <html>
<   <head>
<     <title>This is a Title</title>
<   </head>
<   <body>
<     <p>Hello world!</p>
<   </body>
< </html>
---
> <html>
>   <head>
>     <title>Another Title</title>
>   </head>
>   <body>
>     <p> Bye World</p>
>   </body>
> </html>
```

Listing 5.3: Unix Diff output

Notice how, for a simple example, where the text content of two nodes was changed, the output from the basic tool was far from useful. Now if we apply this to the case in hand we quickly come to realize that this approach was insufficient. The reason, however, is quite simple, a textual diffing algorithm is unaware of the context and structure of HTML, it interprets everything as words instead of nodes, with children and properties.

To tackle this issue we set out to find a diffing tool specifically designed to handle the particularities of HTML. Our requirements were as follows:

1. Come in the form of a library so that we may tweak the code as needed and to use it automatically without third-party dependencies such as Web APIs.

2. Provide a clear and easy to parse output interface so that in the future the adaptive browser's rule generation module may easily interact with it.
3. Uses an algorithm specifically tailored to tree structures as is the case for HTML.
4. Provide corresponding diffing operations such as node updates/moves/inserts and deletes.
5. *Preferred* written in JavaScript so that interoperation with Selenium Webdriver and PhantomJS is simple.

After searching the web for these tools it became apparent that they are currently available in the form of services on the web or through libraries in a variety of languages. However their output is quite simple and not expressive enough in some cases, see listing 5.4 (diff is given in terms of insert/delete operations)

```
1     <del class="diff modified"><title></del><ins class="diff modified"><p class="c2
          "></ins>
```

Listing 5.4: HTML example diff from W3[1]

or hidden behind a web page 5.1 in others. Note that the output provided by the [2]DiffNow web page shows us a somewhat more interesting diff as it has the concept of changed or updated nodes and not merely insert and delete operations:

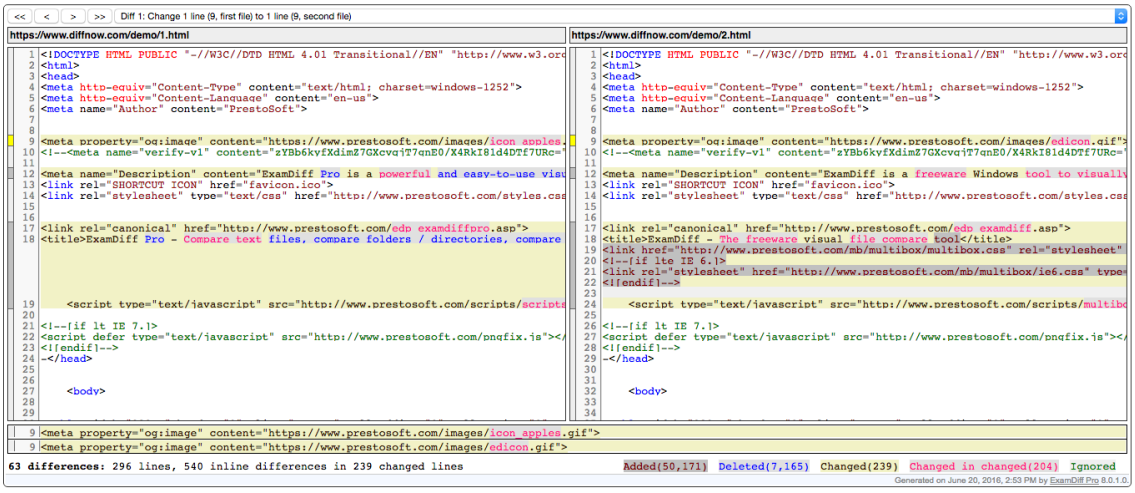

Figure 5.1: HTML Diff service sample output

[1] https://www.w3.org/wiki/HtmlDiff
[2] https://www.diffnow.com/

Eventually, we came to the conclusion that no option was available that fitted all four bases we needed. Instead of settling for one of the available tools and handling incompatibilities later, we decided to create our own diff tool. To do this, we decided to adapt an existing XML diff algorithm published in a paper, Detecting changes in XML documents[12]. The reason behind this choice was, structurally-speaking HTML and XML are similar in their structure as a tree.

The algorithm described in the paper was designed from the ground up with the concept of different types of diff/patch operations, such as element moves and attribute-level updates/inserts and deletes, which is exactly what we were looking for. What's more, we were also able to find an open source implementation of the algorithm for XML using C++ [3]xydiff github that allowed us to see the algorithm in action and especially important to reverse-engineer it to our needs.

5.1 Algorithm

The underlying theory behind creating a tree-based diffing algorithm lies in element-matching. Element matching is how the nodes of two trees are matched together, or in order words how a decision is made to decide which nodes correspond to the "matching" nodes in the second structure. In text-based algorithms, this is simple because the smallest unit for these tools is characters, static representations of data that have only one state. The character "A" will always uniquely match to other characters whose value is "A".

In tree structures the same does not apply, the smallest unit for these is the node. A node is represented by its definition, content (typically an array of alphanumerical value), attributes and more importantly its descendants. Figure 5.2[4] illustrates each of these components for an HTML tag.

Given so many properties for a single node, it becomes obvious that a definition of equivalent nodes is required. What defines two nodes as equivalent changes according to the nature of the tree structure. For XML files for example, there are schemas or structure definition files which specify what characteristics make the node unique. In HTML, the standard has only one equivalent notion, that of the id attribute, it is based on this information that element matching will be done for our HTML diffing tool. After having matched as many nodes between the two trees, the output diff becomes a matter of determining a preferably minimal set of operations that converts the first tree into the second. Non-matched elements are outputed as either inserted or deleted, whilst the output for matched elements depends on the number of properties that are different between them. This

[3] https://github.com/fdintino/xydiff
[4] Retrieved from http://astexplorer.net/

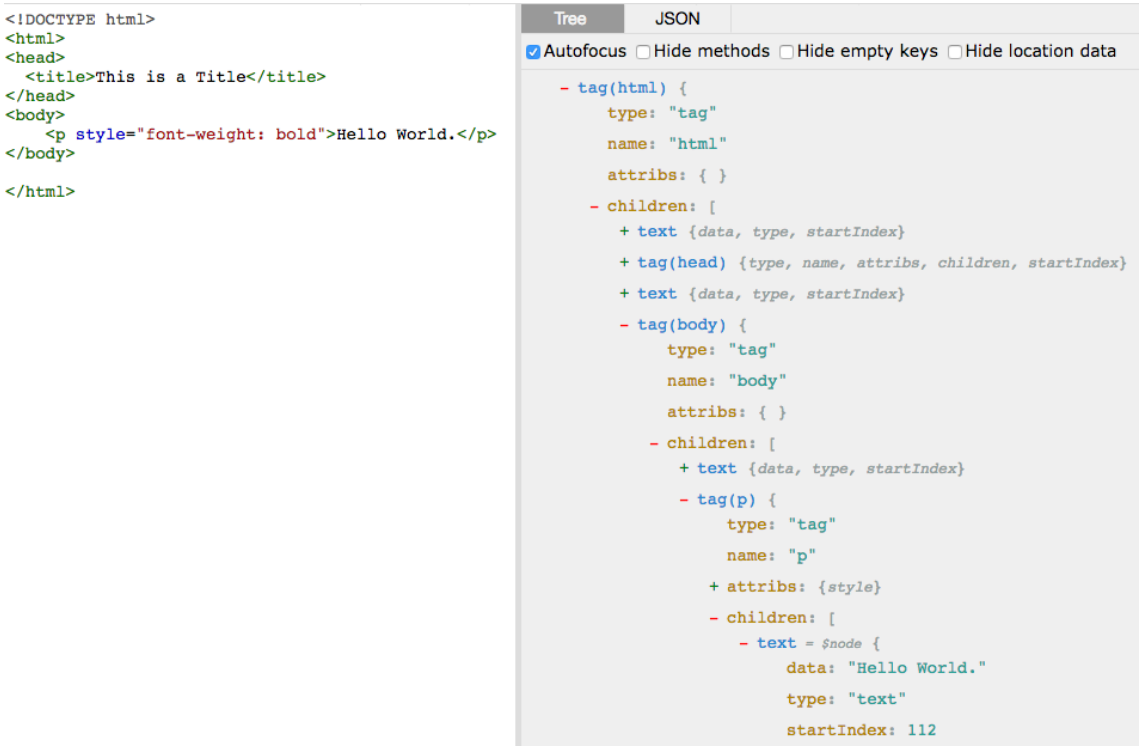

Figure 5.2: AST(Abstract Syntax Tree) representation of an HTML element

way it becomes possible to obtain a very granular level of diff operations: they can be at attribute, content, descendent level.

When initially reviewing the code repository of the XYDiff tool it became evident that the code base was deeply coupled with the XML I/O and parser library Apache Xerces[5]. This meant that reading HTML would not work with the tool because HTML is not as structured nor well-formed (an HTML tag may not require the slash greater than sign to close itself, e.g: <br>). Another breaking point is that the element matching made for XML is not valid for HTML and so incorrect matches would occur.

To handle this problem we took the algorithm described in the paper, which is split into five stages and implemented our tool from the ground up, adapting it where necessary to work with HTML. Before beginning with the algorithm itself the first step was to find an equivalent to the Xerces for HTML, our choice of a library was parse5 [6]. Our choice was made taking into account the compliance with the HTML5 spec to ensure the best compatibility with new HTML tags, the amount of documentation and support provided.

[5] http://xerces.apache.org/
[6] https://github.com/inikulin/parse5

5.1.1 Node traversal and uniqueness retrieval

After parsing both HTML files into AST in-memory objects, the first step is to traverse them. During this traversal uniquely identifiable nodes are searched for, in the particular case of HTML nodes, this means looking for id attributes as referred to above. After this initial pass, the unique nodes are then used to perform direct element matching.

The idea being that if two files being compared have the same origin (one is a modification of the other) and both content elements have the same id, then those are matching points of the files. Imagine for example a button tag with the id "login" which is quite common, if both files present this tag then they match each other. If these nodes are not matched then they will never be matched (assumed that it is an insertion/deletion difference). This stage is particularly useful and critically performant in files with many such unique elements, as most computations are done in this step.

5.1.2 Computation and ordering of node relevance

During the initial pass of both trees, for each node, a signature and weight are calculated. This signature is a hash value (with a very low collision rate, varies from language to language, in our case the md5 hashing function) based on the node's proprieties and that of its children in such a way that this propriety uniquely identifies the node. This signature value works in a similar manner to unique attributes (e.g: id) in that if present in both trees then the two nodes and their respective sub-trees are matched.

The computed weight propriety is used to define the importance of a node in a file. It is the size of the text nodes and the sum of the children node's weights for a non-textual element. Consider, for example, a news website or blog, the most important aspects of these web pages are the textual content they contain. In the HTML representation of one of these web pages, the HTML elements such as titles (<title>), headings (<h1>..<h6>) and paragraphs (<p>) contain the text that is principal value for that page. The more weight a node has on the page, the more crucial to the page it becomes and the more important it is to match.

With this information, sub-trees of the AST (defined by their root node) are ordered in a priority queue, by descending value of weight. Initially, this queue contains only the root of the entire document.

5.1.3 Node matching

From the queue, the root node for the heaviest sub-tree is removed. A list of potential matches for this node in the second document is then created, specifically nodes with the same signature. From this list, the best candidate is selected and matched to the root node that was removed from the queue.

If no match was found (empty list of candidates) and the node is an element (text, comment and directive node's children nodes are discarded), the node's children are added to the queue. A search is then done for that node one level higher, up to a certain value which varies according to the weight of the node.

If the match list contains more than one element, the best candidate is the one whose parent's reference matches the reference candidate node's parent. When a candidate is found, we match the subtrees and ancestors of the nodes if their labels are the same. Again the number of ancestor pairings done varies according to the weight of the node to give the greatest match to the heaviest sub-tree.

5.1.4 Optimizations

This stage is done to pair previously unmatched nodes where their respective children or parent nodes are already matched. To do this two traversals are done, first in a bottom-up and then top-down fashion. The main advantage of this stage is to improve the quality of the diff produced and to reduce the number of computations done in the final stage.

This method assumes the following: For a given node I that is not matched, but one of its children C is paired to C', the algorithm will match I to the parent node of C'. The parent of the paired node selected is the largest (weight-wise) of the list available C'1, C'1, ..., C'n. If a node is paired but both contain a unpaired node with the same label, they will be matched. Considerations about this stage are given in the original paper.

5.1.5 Diff computation

The fifth and final stage of the algorithm is to create the diff operations from the tree-matching previously calculated. Inserted or deleted nodes are easily calculated as they would have no matched node. Moved operations are also easy to compute by checking whether parent nodes are matched or not if two nodes are matched but their parents are not, then subtree move was found. Attribute

and content-level deltas are also easy to detect, the first would be to check if an attribute is detected in one node and not another. The content and attribute value are both calculated through simple textual diff between the version. The difficulty in this stage is to detect changes in node order for a subtree. To solve this problem, a longest common subsequence approach was suggested by the original paper but only for subtrees containing a maximum of 50 immediate children, which in the context of HTML documents for our use case was a reasonable value.

5.2 Output

So that the presentation of the diff data shown in the next chapter can be better understood[7], we present here a sample of a diff output generated from our tool. A comparison between two HTML files will be given in XML format, where each element corresponds to a patch operation that converts one document into the other. One very important aspect to note is the concept of xid or node identifier. This is the post-order identification of the node in the first tree that converts into the second. The basic structure for the operation is as follows:

- Attribute level operations:
  - Attribute inserted: <ai a="attribute id" v="inserted value" xidReference/>
  - Attribute updated <au a="attribute id" nv="new value" ov="old value" xidReference/>
  - Attribute deleted <ad a="attribute id" v="removed value" xidReference/>
- Node level operations:
  - Element inserted: <i par="parent xid" pos="position in parent's child list" xm="xid of inserted subtree"/>
  - Element updated: <u par="parent xid" pos="position in parent's child list" oldxm="old xid value"> text level operation:
    1. text inserted: <ti pos="position of operation in string">Data</ti>
    2. text removed: <tr pos="position of operation in string" len="number of characters to delete"/>

    </u>
  - Element removed: <d par="parent xid" pos="position in parent's child list" xm="xid of removed subtree"> subtree </d>

An example of the output of the diff tool is shown in Listing 5.5.

[7] A detailed explanation of the structure and theory behind it, are given in [12].

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xy:unit_delta xmlns:xy="urn:schemas-xydiff:xydelta">
   <xy:t> <!-- Start of transformation list of operations -->
      <!-- Attribute level operations -->
      <xy:ai a="manifest" v="demo.url.com" xid="25"/> <!-- attributed inserted -->
      <xy:au a="placeholder" nv="email" ov="username" xid="16"/> <!-- attributed
          updated -->
      <xy:au a="type" nv="password" ov="text" xid="17"/>
      <xy:au a="placeholder" nv="palavra passe" ov="email" xid="17"/>
      <xy:ad a="class" v="login_bt" xid="20"/> <!-- attributed deleted -->
      <xy:au a="type" nv="text" ov="password" xid="18"/>
      <xy:au a="placeholder" nv="username" ov="palavra passe" xid="18"/>
      <!-- Element level operations -->
      <xy:d par="22" pos="3" xm="19" move="yes"/>
      <xy:u par="14" pos="1" oldxm="13">
         <xy:tr pos="0" len="1"/> <!-- child nodes of element-level operations
             define a series of content-level patch operations to execute-->
         <xy:ti pos="0">Ag</xy:ti>
         <xy:tr pos="4" len="2"/>
         <xy:tr pos="5" len="3"/>
      </xy:u>
      <xy:d par="9" pos="1" xm="8"><div class="front_clip"><div class="ring_clip"><
          /div></div></xy:d> <!-- element deleted -->
      <xy:i par="15" pos="3" xm="(57-58)"><p>por favor</p></xy:i> <!-- element
          inserted -->
      <xy:i par="15" pos="4" xm="(59-61)"><div><p>algum texto de teste</p></div></
          xy:i>
      <xy:i par="22" pos="4" xm="23" move="yes"/>
   </xy:t>
</xy:unit_delta>
```

Listing 5.5: HTML diff output from created tool

Chapter 6

Results

Having discussed the design of our system and the tools to be used, we will present in this chapter the tests and results for the two variables of our work. Namely, the performance criterion for the base browser of the system, first mentioned in chapter four and the output generated from our diffing tool when comparing the outputs of these tests.

6.1 Browser performance

The first thing we sought to discover was which of the browser technology options, PhantomJS or Selenium was the better option performance-wise, for our adaptive system. For this particular experiment, the browsers used were Google Chrome (v51.0) and Mozilla Firefox (v46.0). Internet Explorer, Safari were discarded because of their dependency on the operating system used. The Opera browser although works in multiple operating systems, was also discarded because it uses the same render engine as Google Chrome (Blink). Other browsers were considered to have too small a usage percentage-wise.

6.1.1 Test Structure

In order to remove biased and increase impartiality, the tests were run 10 times. Averages were then calculated and used to generate the data presented in 6.1.2. The performance test setup was done as described below:

1. For every tag defined in W3C.

(a) For every browser technology available in the system.

i. Start timer the tag.

ii. Open the corresponding test.

iii. Wait for the browser to finish loading and processing the page.

iv. Download the processed DOM through JavaScript code:

```
1                document.querySelector('html').outerHTML;
```

v. Stop timer and record the time taken for the tuple <browser,tag>

(b) Repeat 10 times.

6.1.2 Data

Because of a large number of existing HTML tags (over 100) that were tested, it was decided to group certain tag-related tests by category[1]. These categories define the common functionality that each tag in the group is used for. The data that follows is presented in the form of box plots, where the processing times for each browser is shown. The head and tail of each set define the 0-25th and 75-100th percentile with the bulk of the results]25-75[values, being displayed in the colored box. This is to showcase the performance range of particular groups of tests which may have taken long or shorter compared to the rest. The original data can be viewed in the appendix section.

6.1.2.1 Basic HTML

In order to generate the data presented below, the following tags were considered: <title>, <body>, <h1> to <h6>, <p>, <br>, <hr> and <comment>. These tags were grouped together in part because of their existence since the beginning of HTML, and used to exemplify the most basic of HTML files (the notorious Hello World example).

In the data presented by Figure 6.1 PhantomJS shows the variation between the fastest and slowest render, being of 0.06 seconds between all tag renderings, with the 25-75 percentile contained within a 0.01 second range and executing on average in 0.67 seconds. Chrome presents an average of 2.10, range of 0.96 and 25-75 range of 0.03 seconds. Firefox reports an average of 3.81, variation of 1.14 and 25-75 percentile range of 0.93 seconds. There is an increase in the range of processing time by a factor of 18.3 between PhantomJS and Firefox.

[1] As shown here http://www.w3schools.com/tags/ref_byfunc.asp

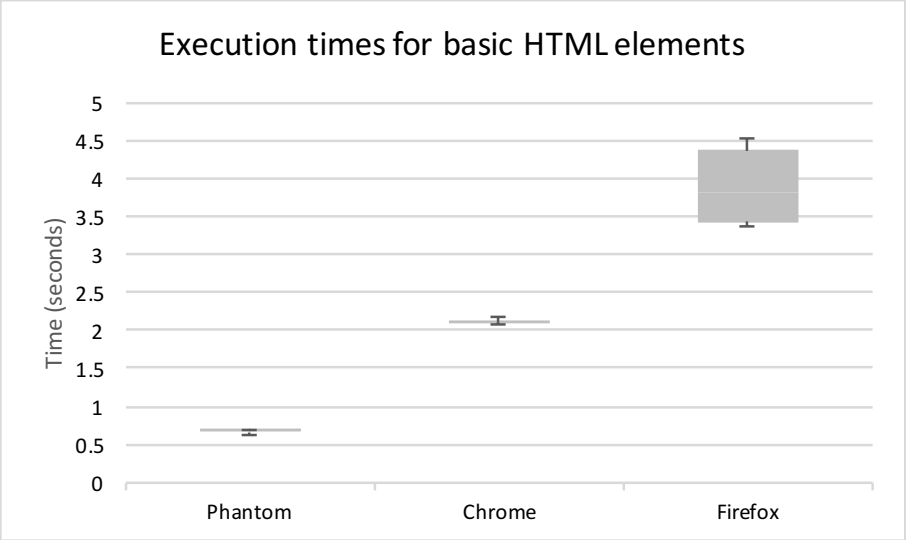

Figure 6.1: Performance graphic for basic HTML element rendering

6.1.2.2 Formatting HTML

Formatting HTML refers to all elements which in some way provide specific presentation for the content they contain. These are: <abbr, <address>, <b>, <bdi>, <bdo>, <blockquote>, <cite>, <code>, , , <samp>, <var>, <kbd>, <del, <dfn>, <i>, <ins>, <mark>, <meter>, <pre>, <progress>, <q>, <rp>, <rt>, <ruby>, <s>, <small>, <sub>, <sup>, <time, <u> and <wbr>.

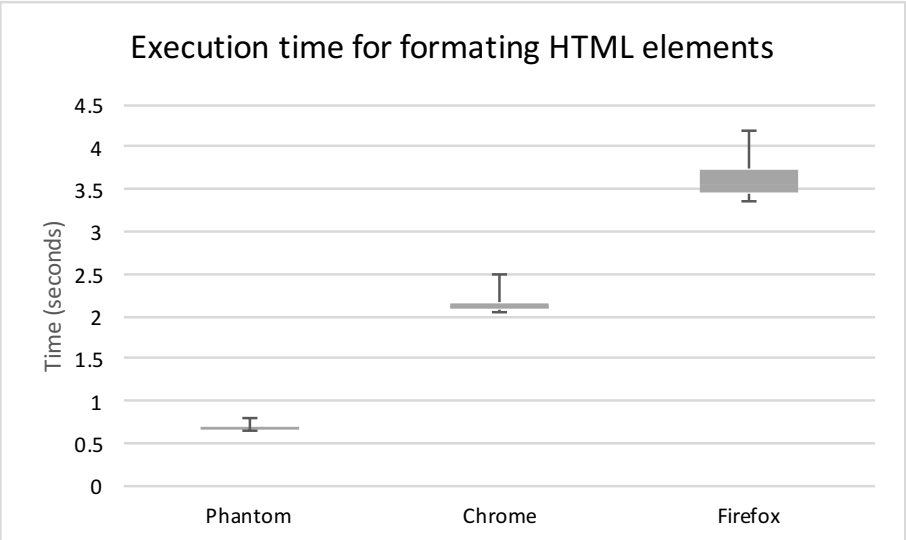

Figure 6.2: Performance graphic for format-related HTML element rendering

Figure 6.2 shows that PhantomJS executes the group of elements on average in 0.67 seconds,

a range of 0.13 and 25-75 range of 0.02 seconds. Chrome averages at 2.10, a range of 0.43 and 25-75th percentile variance of 0.07 seconds. Firefox takes 3.55 seconds on average per execution, with a range of 0.82 and 27-75 variance of 0.29 seconds. There is a factor 6.11 between PhantomJS and Firefox's execution ranges.

6.1.2.3 List and Table HTML

This group of elements are defined by their use in representing cluster of related information, be it by list, enumerations or tables. The elements that compose this group are: <li>, <ol>, <ul>, <dl>, <dt>, <dd>, <menu>, <menuitem>, <col>, <table>, <tr>, <th>, <td>, <caption>, <tbody>, <thead>, <tfoot>, <colgroup>.

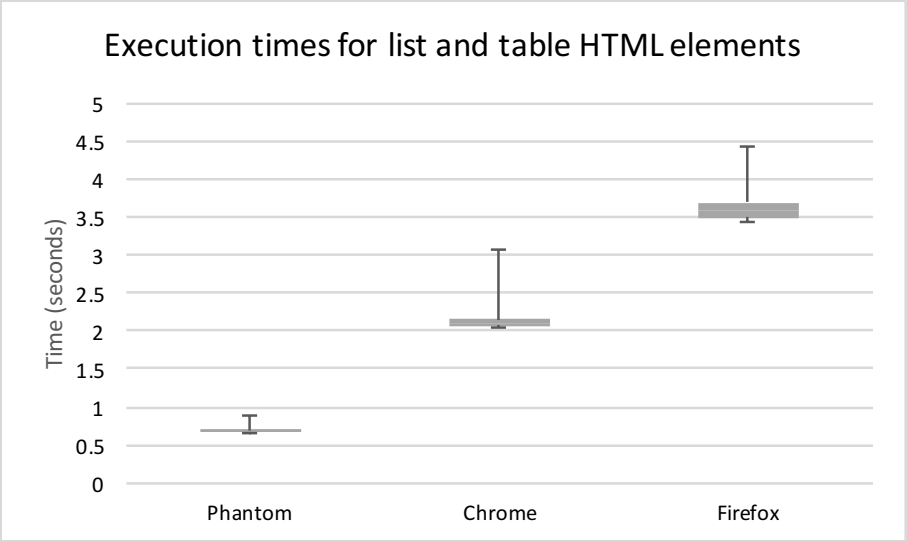

Figure 6.3: Performance graphic for list and table HTML element rendering

Figure 6.3 reports: PhantomJS averaging 0.67, a range of 0.22 seconds and 25-75th percentile variance of 0.02 seconds; Chrome on average takes 2.10, ranges by 1.01 seconds between the minimum and maximum, and varies in 25-75 range by 0.07 seconds; Firefox required 3.60 seconds on average, ranged by 0.97 and contained the 25-75 values in a 0.21 second range. There was a reported factor of 4.33 between the ranges of PhantomJS and Firefox.

6.1.2.4 Form and Input HTML

When using web pages, the most distinguishable elements a user sees are the ones he also interacts with. These elements provide the essential basis that give websites their dynamism. This group is composed of the following tags: <form>, <input>, <textarea>, <button>, <select>, <optgroup>, <option>, <fieldset>, <legend>, <datalist>, <keygen>, <output>.

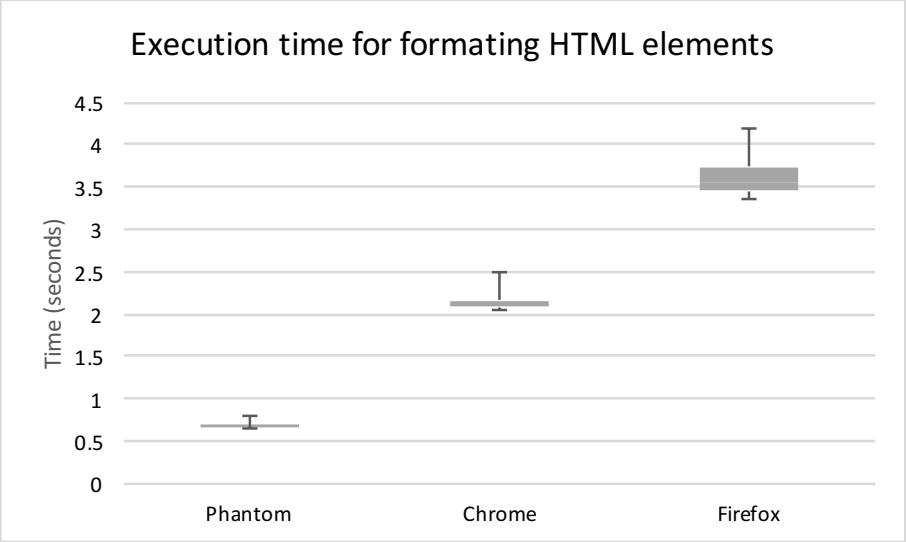

Figure 6.4: Performance graphic for form and input HTML element rendering

Figure 6.4 shows PhantomJS averaging 0.68, ranging 0.08 and varying between the 25th and 75th percentile by 0.01 seconds. Chrome averaged 2.14, ranged 0.23 and contained the scope]25,75[in a 0.05 second range. Firefox took on average 3.66 seconds, ranged by 0.91 and had a 25-75 variance of 0.27 seconds. The factor of ranges between Chrome and Firefox was of 11.5.

6.1.2.5 Dynamic HTML

This group is comprised mostly of relatively new elements defined in the HTML5 specification. They are elements which provide developers with the ability to reference interactive external resources such as video and audio. They are: <iframe>, <audio>, <source>, <track>, <video>.

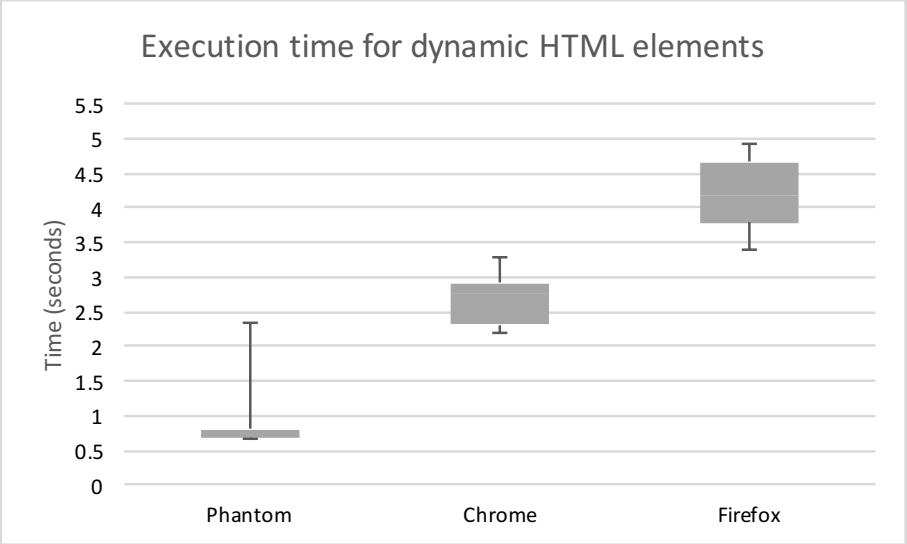

Figure 6.5: Performance graphic for dynamic HTML element rendering

Figure 6.5 details that PhantomJS averaged 0.67 seconds, varied 1.68 between faster and slowest render, and the range 25-75 was contained in a 0.124 second interval. Chrome averaged 2.75, ranged 1.10 and had the 25-75 range in 0.6 second interval. Firefox averaged 4.17 seconds, 1.52 between $[0\text{-}100]^{th}$ percentile and 0.87 second variance in the]25-75[range. There was also a reported factor 0.9 (10% percent slower) range difference between the PhantomJS and Firefox renders.

6.1.2.6 Styles and Semantics HMTL

The following group is characterized by elements which define a layout for a specific document:<style>, , <header>, <footer>, <main>, <section>, <article>, <aside>, <details>, <dialog>, <Summary>.

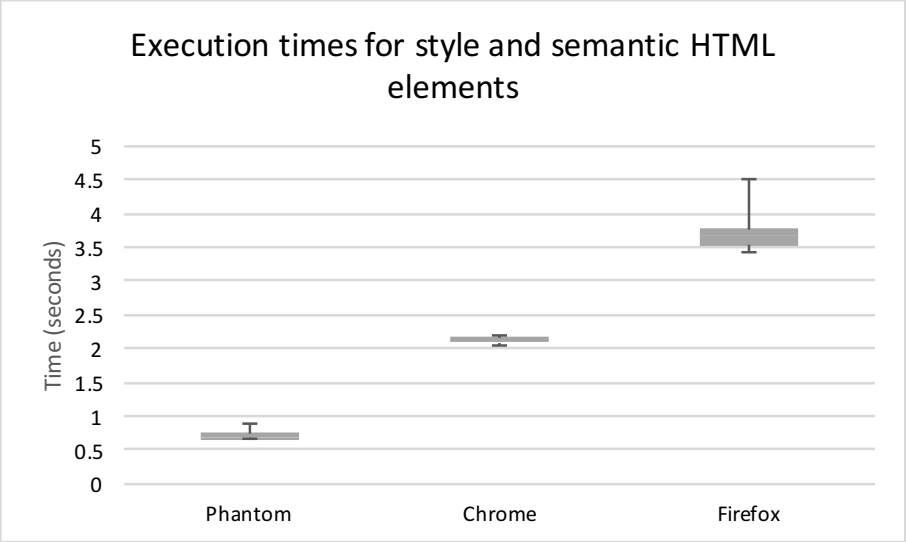

Figure 6.6: Performance graphic for style and semantic HTML element rendering

Figure 6.6 reported PhantomJS averaging 0.68, ranging 0.22 and variating by in the 25-75th range by 0.08 seconds. Chrome executed on average in 2.11, ranged by 0.15 and ranged in the 25-75 spectrum by 0.08 seconds. Firefox averaged by 3.68, ranged by 1.10 and varied by 0.23 seconds in the 27-75 series. PhantomJS and Firefox's range values varied by a factor of 5.1.

6.1.2.7 Images and Links HTML

In order to provide a pleasure visual experience to the user, the browser allows images to be inserted. Frequently these are done by referencing the URL where the resource is located. To reference these external or internal resources links are used. This group is consequently composed of elements that enable these connections (there are others by they are grouped by a more distinct characteristic): <image>, <a>, <nav>

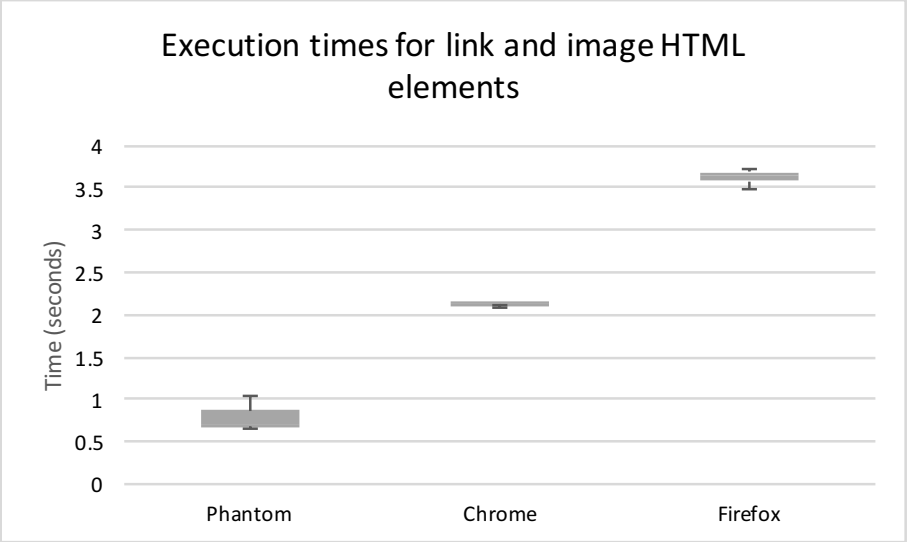

Figure 6.7: Performance graphic for image and link HTML element rendering

Figure 6.7 announced the following values: PhantomJS average as 0.70, range as 0.38 and]25-75[as 0.19 seconds. Chrome's values for processing times, were 2.11 seconds on average, 0.05 seconds in the [0-100] range and 0.02 seconds in the]25-75[range. Firefox averaged 3.65, ranged 0.21 and varied in the]25-75[th percentile by 0.11 seconds. The recorded range factor difference between PhantomJS and Firefox was of 0.6.

6.1.2.8 Meta and Programming HMTL

The final group of elements pertain to meta-information, useful for the browser to provide non-standard functionality to the user, typically by the developer of the web page. Their purpose is to provide a bridge between an HTML document and custom JavaScript functionality. These elements are: <head>, <meta>, <base>, <script>, <noscript>, <embed>, <object>, <param>.

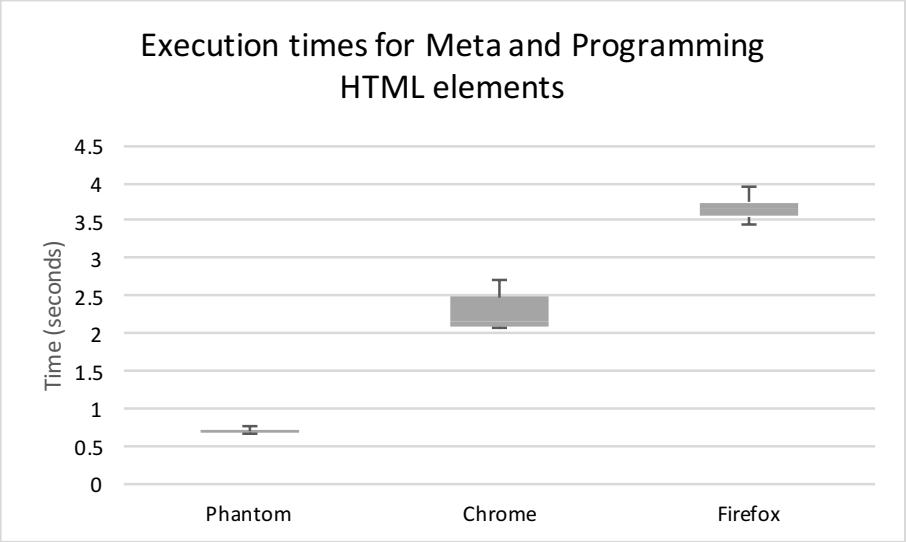

Figure 6.8: Performance graphic for meta and programming HTML element rendering

Figure 6.8 shows PhantomJS averaging 0.68, ranging 0.13 and varying between the 25th and 75th percentile by 0.02 seconds. Chrome averaged 2.16, ranged 0.65 and contained the scope]25,75[in a 0.40 second range. Firefox took on average 3.66 seconds, ranged by 0.51 and had a 25-75 variance of 0.19 seconds. The factor of ranges between Chrome and Firefox was of 4.1.

6.2 Cross-browser DOM Consistency

6.2.1 Test Structure

The second part of our experiment was to evaluate the differences between the generated DOM of the different browsers. We aimed to look for tags whose processed DOM was different in other browsers. The browser technologies evaluated and testing conditions in these tests were the same as the performance tests. For this test, the same browsers were used. Adding Internet Explorer to the list of browsers was at first a possibility, but during testing, its Selenium webdriver appeared to not work with our setup. Further investigation into this issue showed that the IE browser does not support access to local files through the file:/// schema[2] without considerable tweaking which also impacted the access of that file to external resources. This meant that testing elements such as <images> with an URL reference to third-party hosts locally would not work in IE.

The testing process setup was as follows:

[2] https://goo.gl/rvCWw6

1. For every tag defined in W3C.
   (a) Open the corresponding test locally in every browser technology.
   (b) Wait for the browser to finish loading and processing the page.
   (c) Download the processed DOM through the following JavaScript code:

```
1    document.querySelector('html').outerHTML;
```

   (d) Store result of the JavaScript query.
2. Compare processed data with output generated by the base browser, using diff tool.
3. Evaluate diff output.

6.2.2 Data

Considering the number of combinations between the HTML tags and browsers evaluated(3*80^{+}), we will not detail here all the results of this test suite. Many of the tests created generated no difference between the browsers considered. Hence we will limit our report, to only noting the diff outputs which generated differences between browser renders. There was one consistent difference found between Firefox's renders compared to the rest. Every single rendered file had the attribute webdriver="true" added to the <html> element node. This attribute had no detected impact in the rendered DOMs. The following tags generated significant data to analyze[3]:

1. <bdi>
2. <iframe>
3. <keygen>
4. <rp>,<rt>,<ruby>. These three elements were tested together because they were so specified, in the original sample code[4].
5. <track>

Of these, a few were found to be inaccurate. The <bdi> element test generated diff output with content from all browsers technologies, when compared to the original test. However this diff data is consistent throughout all the browsers, meaning that cross-browser no difference was detected,

[3]Given authorization, the tests and result files for these cases will be made available in a zip file for peer-review
[4]http://www.w3schools.com/tags/tag_rp.asp

it was merely a consistent processing of the element. The <iframe> test showed no significant differences other than an apparent encoding option done by the WebKit engine for its content. The rendering engine converts the "<" and ">" characters into "<" and ">" respectively. This may be due to the way the engine handles the interpretation of the iframe's content property. The <rp>,<rt>,<ruby> test showed that all browsers make an underlying change to the encoding of the content of these nodes. No more information could be retrieved from this particular experiment.

The <keygen> element in the Firefox engine when processed is replaced by a <select> tag, additionally this browser also explicitly creates two <option> child elements specifying a default behavior for the <keygen>. This behavior was not found in either the Chrome or PhantomJS outputs. With regards to the <track> element, we detected that PhantomJS does not support it because it's output is identical to the original test but does not contain the DOM information for the URL specified. The reason behind this lack of support is that the developers behind this technology chose to not implement the underlying necessary codecs[5]. Both Google Chrome and Mozilla Firefox adequately processed this tag and no difference between their outputs was found.

6.3 Discussion

The figures presented in 6.1.2 show a recurring perception of the performance of the three browsers. The initial assumption that headless browser technologies like PhantomJS are more performant than the typical browser seems validated. The overhead of running a UI and the underlying cost of displaying an interface appears to be quite significant. Even when considering the faster Chrome browser, there is still a noticeable difference in performance between headless and non-headless technologies.

However, this apparent performance of PhantomJS comes with a few nuances. When looking at specific groups of HTML, the dynamic HTML group of data 6.4 reveals an interesting discrepancy. It is the only set of elements that contradict the general perception of the performance of PhantomJS. The particular test that generated this viewpoint was the <iframe> test. When analyzing the content against the outputs of the other browsers, no differences were found in the output between the original test and that of the output of PhantomJS. This leads us to the hypothesis that when a web page contains links to external resources via this element, the rendering performance seen in PhantomJS is counterbalanced by optimizations made by browsers in other areas, such as network. Additionally as mentioned 6.2.2, HTML5 dynamic elements like <track> are not supported. This means that PhantomJS does not request the information specified by the content attribute nor the additional content downloaded from the URL, explaining the apparent performance for these tests.

[5]See here for more information http://phantomjs.org/supported-web-standards.html

An interesting conclusion about the traditional browsers was also made. Google Chrome consistently outperforms Firefox, both in boot-up time and element processing. The evidence suggests an overall lack of optimization, in the Gecko engine for DOM processing, that was found in the Webkit engine (PhantomJS uses Webkit and Chrome a fork of it, Blink).

Although the set of information that was able to be retrieved in limited, it allows us to obtain some useful conclusions which characterize the browsers being analyzed. Due to the complicated architecture of browsers and the variety of elements, it is quite possible that the same type of tests for all elements does not exercise them equally. Chapter 7 provides some details on improvements in this area. However from our tests, a transformation rule as was initially described in chapter four, appears to exist. That is, for every <keygen> element, if the base engine is WebKit and the target browser is Firefox the following transformation should be done: Replace any <keygen> element with a <select> element, preserving the attributes of the original tag. Two child elements of type <select> and values high and medium respectively should be added.

With regards to the choice of browser technology to be used in the Adaptive Browser idea also presented in chapter 4 the choice is Google Chrome. The reasoning being that, for such a system to correctly handle a typical, "out-in-the-wild" HTML file, it must support all functionalities that exist for HTML files. This means that phantomJS must be discarded, despite its performance advantage over traditional browsers. If at any point, a headless technology with similar performance and full implementation of all elements arise, it would be simple to exchange the base browser of the system, requiring only that the system learn the corresponding transformations.

Chapter 7

Conclusion

We have proposed a new approach to cross-browser rendering and created some basic tooling which enabled us to validate our proposal. We devised a testing framework which has enabled us to characterize the differences, both in terms of performance and correctness of HTML rendering, between normal vendor browsers such as Mozilla Firefox and Google Chrome, and headless browser technology specifically PhantomJS. To enable us to better analyze the results of this framework, we created an HTML diffing library that enables a user to analyze the content of two HTML files, whilst disregarding differences commonly caught in standard diff utility tools.

From our experimentation, we have evidence to conclude that PhantomJS is a good automation tool where performance in standard browsers is inadequate. PhantomJS' HTML rendering boasts of consistently greater performance than all other browsers considered. However, this tool is also limited in its use. More complicated features such as <video>, <audio> and <track> may not be correctly processed, inducing incorrect results. Our recommendation is then to use phantomJS where an intimate knowledge of the web page to be automated is known beforehand and where the aforementioned unsupported elements are not used. For other situations Selenium is the adequate solution has it brings extra compatibility, security and robustness that comes with using traditional browsers.

Our second part of experimentation validated our hypothesis that browsers are inconsistent at a DOM-level. While this inconsistency is not as extensive as initially thought, it will continue to exist, partly due to constant browser updates and ever-changing HTML specifications. These results show that further investigation and tuning of the tests used is required before validating the theory of creating a system capable to emulating browsers based on HTML, CSS and JavaScript unitary transformation rules.

7.1 Future Work

During our experiments, we identified some potential areas of improvement. In regards to the diffing algorithm, some improvements could have come in the form of HTML-specific element matching. For example matching elements with static attributes such as urls and href for scripts, iframes, stylesheets and video. By searching the tree for the existence of these elements and match by their urls a better element matching could be done. This would allow us to reduce the impact of the heuristics of the original algorithm regarding parent and child matchings of elements-matched nodes. Other more recent algorithms have been proposed [34]. The X-diff method reports improved subtree matching which leads to optimal solutions, albeit at a performance cost. For our experiment performance costs can be tolerated for the rule generation process, which may improve the overall data set for non-supervised learning.

One of the limitations mentioned for our diff tool was the HMTL parser used, which fixes malformed HTML files. Replacing this library with a parser which does not attempt to correct the file would allow better detection of incorrect implementations of the parsers used by browsers. A related enhancement to the diffing tool would be to modify its output, so as to make it easier for rule-deduction. From a human-readability point-of-view, the use of XIDs is not very useful to reason about a difference, using XPath expressions instead could prove effective.

In regards to our DOM tests for inducing transformation rules. The completeness and thoroughness of the tests used should be revised. When testing a certain element, an important aspect is to see the processed DOM output under multiple conditions. For instance, to identify how does a browser fix broken tags, malformed syntax, missing mandatory attributes or incorrect child elements for a specific element. Defining tests that answer these questions can help in identifying particularities in the rendering engines. For this a look into Web platform tests[1] is a good starting point to improve the quality of the tests.

It is clear that a lot of work is still needed in order to have the necessary basis for an adaptive browser. The improvements mentioned above list some potential approaches that could lead to a better understanding of the differences between browsers. In addition, approaches on how to understand how CSS and JavaScript function in multiple browser environments are necessary.

[1] W3C has a suite of tests available at https://github.com/w3c/web-platform-tests

References

[1] Blink - the chromium projects. Available at http://www.chromium.org/blink.

[2] Detecting html5 features. http://diveintohtml5.info/everything.html.

[3] Internet explorer driver server. Available at http://www.seleniumhq.org/download/.

[4] Open source web browser engine. Available at https://webkit.org/.

[5] Respond to your user's browser features. https://modernizr.com/.

[6] A scriptable browser for web developers. Available at https://slimerjs.org/.

[7] Awesomium. Global javascript object creation. http://wiki.awesomium.com/tutorials/tutorial-3-hooking-up-events.html.

[8] G. W. C. Blog. A proposal for making ajax crawlable. https://googlewebmastercentral.blogspot.com.au/2009/10/proposal-for-making-ajax-crawlable.html.

[9] W. Chou. Web services: Software-as-a-service (saas), communication, and beyond. In *Congress on Services Part II, 2008. SERVICES-2. IEEE*, pages 1–1, Sept 2008.

[10] G. I. o. T. A. G. U. . P. M. . O. A. Choudhary, S.R. ; Coll. of Comput. Crosscheck: Combining crawling and differencing to better detect cross-browser incompatibilities in web applications. *Fifth International Conference on Software Testing, Verification and Validation*, April 2012.

[11] S. Choudhary, H. Versee, and A. Orso. Webdiff: Automated identification of cross-browser issues in web applications. In *Software Maintenance (ICSM), 2010 IEEE International Conference on*, pages 1–10, Sept 2010.

[12] G. Cobena, S. Abiteboul, and A. Marian. Detecting changes in xml documents. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 41–52. IEEE, 2002.

[13] W. W. W. Consortium. https://www.w3.org/TR/html5/.

[14] W. W. W. Consortium. Html element reference. Available https://www.w3.org/TR/html-markup/elements.html.

[15] W. W. W. Consortium. Html standards and revisions. Available at https://www.w3.org/TR/#tr_HTML.

[16] D. . Curran. Man in the browser attacks. *International Journal of Ambient Computing and Intelligence*, pages 29–39, January 2012.

[17] O. Dictionaries. http://www.oxforddictionaries.com/definition/english/browser.

[18] M. Foundation. Gecko - mozilla | mdn. https://developer.mozilla.org/en-US/docs/Mozilla/Gecko.

[19] A. Grosskurth and M. Godfrey. A reference architecture for web browsers. *Proceedings of the 21st IEEE International Conference on Software Maintenance*, pages 661 – 664, September 2005.

[20] html5test. Html5 compliance test. https://html5test.com/compare/browser/mybrowser/firefox-40/operamobile-31/ie-11/ie-Edge%2013.html.

[21] A. Mesbah and M. R. Prasad. Automated cross-browser compatibility testing. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 561–570, New York, NY, USA, 2011. ACM.

[22] M. Organization. Firefox releases. Available at https://www.mozilla.org/en-US/firefox/releases/.

[23] PhantomJS. headless webkit scriptable with a javascript api. Available at http://phantomjs.org.

[24] Phantomjs. Supported web standards. http://phantomjs.org/supported-web-standards.html.

[25] S. Roy Choudhary, M. R. Prasad, and A. Orso. X-pert: Accurate identification of cross-browser issues in web applications. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 702–711, Piscataway, NJ, USA, 2013. IEEE Press.

[26] S. Roy Choudhary, M. R. Prasad, and A. Orso. X-pert: A web application testing tool for cross-browser inconsistency detection. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, ISSTA 2014, pages 417–420, New York, NY, USA, 2014. ACM.

[27] U. K. Schulz and S. Mihov. Fast string correction with levenshtein automata. *International Journal on Document Analysis and Recognition*, 5(1):67–85, 2002.

[28] Selenium. Selenium webdriver. http://docs.seleniumhq.org/docs/03_webdriver.jsp#how-does-webdriver-drive-the-browser-compared-to-selenium-rc.

[29] SeleniumHQ. Browser automation. Available at http://www.seleniumhq.org/.

[30] I. W. Stats. World internet usage and population statistics. Available in http://www.internetworldstats.com/stats.htm, November 2015.

[31] S. G. Stats. Top 5 desktop, tablet console browsers. Data available at http://gs.statcounter.com/).

[32] W3Techs. Usage of client-side programming languages for websites. Available at http://w3techs.com/technologies/overview/client_side_language/all.

[33] W3Techs. Usage of javascript libraries for websites. http://w3techs.com/technologies/overview/javascript_library/all.

[34] Y. Wang, D. J. DeWitt, and J.-Y. Cai. X-diff: An effective change detection algorithm for xml documents. In *Data Engineering, 2003. Proceedings. 19th International Conference on*, pages 519–530. IEEE, 2003.

[35] Q. WebKit. Webkit port on top of qt. Available at https://wiki.qt.io/Qt_WebKit.

REFERENCES

Appendices

Appendix A

Phantom execution times

file	iteration 1	iteration 2	iteration 3	iteration 4	iteration 5	iteration 6	iteration 7	iteration 8	iteration 9	iteration 10	average
a.html	0,837262190	0,666479508	0,636273177	0,643596424	0,864864316	0,663374271	0,649508559	0,647265963	0,679712095	0,671271905	0,695960841
abbreviation.html	0,860680415	0,830433644	0,640196912	0,647869485	0,915636276	0,729796775	0,765445509	0,807688864	0,740325844	0,756552471	0,769462620
address.html	1,024156179	0,700652210	0,745140600	0,911470845	0,734363184	0,690829626	0,871818595	0,911574859	0,637216291	0,638579462	0,786580185
area.html	0,803241228	0,658212210	0,649096796	0,676104109	0,642272422	0,636130838	0,669858047	0,660724411	0,653835507	0,677096485	0,672657205
article.html	0,926698882	0,635650145	0,795445125	1,238155788	0,723536740	0,920102062	0,744606146	0,741171992	0,806003840	0,820576362	0,835194708
aside.html	0,864648277	0,645801392	0,638470855	0,642439761	0,687084360	0,828449664	0,918365242	0,690127471	0,711007935	0,794330007	0,742072496
audio.html	0,679266029	0,867329873	0,904845052	0,857147885	0,640494422	0,837041829	0,967689533	0,897420138	0,671734782	0,645335837	0,796830538
base.html	0,716009710	0,722389265	0,808162673	0,727973849	0,699769649	0,717488598	0,761652417	0,845807468	0,905399064	0,828785349	0,773343804
bdi.html	0,643034207	0,826824327	0,975836278	0,649641093	0,684341025	0,659256964	0,644353630	0,632039139	0,633103799	0,651185351	0,699961581
bdo.html	0,613380123	0,693337680	0,762816962	1,034306403	0,640073462	0,647316641	0,654900910	0,683086333	0,653788931	0,633953349	0,701696079
blockquote.html	0,646838752	0,634492196	0,638469841	0,634106473	0,660606246	0,647362433	0,655833448	0,847012884	0,652937742	0,646753942	0,666441396
body.html	0,649531463	0,699696106	0,668442676	0,650498313	0,696392676	0,659516180	0,641659499	0,668921386	0,636180233	0,736454925	0,670729346
bold.html	0,642505348	0,797603730	0,650243724	0,653698132	0,654892221	0,666466693	0,640620714	0,690154687	0,658849955	0,670022477	0,672505768
breakline.html	0,659530871	0,645947513	0,643154211	0,664078262	0,648504765	0,668052679	0,824992103	0,637830056	0,669879337	0,675033881	0,673700368
button.html	0,674841443	0,682562043	0,665342473	0,653092721	0,645875853	0,690063555	0,692998902	0,660044200	0,664146575	0,653015450	0,668198322
canvas.html	0,823824334	0,635293684	0,658560164	0,672940382	0,646767542	0,652708202	0,651999384	0,690310111	0,651706209	0,658481409	0,674259142
caption.html	0,650730174	0,642598137	0,657892231	0,654480988	0,807748406	0,670834353	0,667815338	0,651035784	0,778710744	0,675853441	0,685769960
cite.html	0,671036309	0,676209039	0,655215333	0,675385209	0,645513894	0,657823089	0,672157295	0,790708119	0,653547636	0,641529011	0,673912493
code-em-strong-samp-var-kbd.html	0,689441461	0,640796487	0,640909726	0,665495083	0,649241134	0,702993024	0,632461522	0,658811305	0,640694276	0,647025072	0,656786909
col-table-tr-th-td.html	0,797159249	0,646878122	0,665367984	0,639508243	0,645349729	0,660013263	0,643664090	0,654976802	0,651248332	0,677142394	0,668130821
colgroup.html	0,666264338	0,648047791	0,649161038	0,668596177	0,704193567	0,766043181	0,651007583	0,647662001	0,656930170	0,674659359	0,673256521
comments.html	0,631391720	0,629521424	0,632622606	0,646530325	0,627108055	0,620115507	0,657306784	0,630705251	0,624741004	0,632505338	0,633254801
datalist.html	0,833102015	0,666443841	0,674576571	0,678316595	0,679453031	0,679920603	0,687875823	0,659921568	0,653420746	0,671062297	0,688409309
dd.html	0,738791887	0,660447117	0,630719449	0,729654118	0,752871917	0,654597079	0,638651488	0,639726460	0,659088631	0,651303259	0,675585141
del.html	0,657403067	0,644936965	0,656337318	0,653644461	0,698028248	0,641627888	0,653995395	0,673965727	0,697624180	0,652954920	0,663051817
details.html	0,685954747	0,669735384	0,644919282	0,679765330	0,657713648	0,638757456	0,644848200	0,651501538	0,650823078	0,648781991	0,657280065
dfn.html	0,647610867	0,733107560	0,679257363	0,790283603	0,743767537	0,624389392	0,627293844	0,657582102	0,623776178	0,656021732	0,678309018
dialog.html	0,623525584	0,638791609	0,644672821	0,621967635	0,624175880	0,792221923	0,821387654	0,670420915	0,627016803	0,637660247	0,670184107
div.html	0,659819890	0,623631859	0,637349631	0,669277000	0,628318728	0,624910220	0,637581256	0,637362066	0,627954623	0,617920420	0,636412569
dl-dt-dd.html	0,670833137	0,765822292	0,628233482	0,628763955	0,631452042	0,640407721	0,637985555	0,630069316	0,652838798	0,666304356	0,655271065
embed.html	0,651876417	0,631240328	0,648291322	0,666131778	0,627021981	0,669353225	0,890405781	0,645445948	0,631000107	0,634186707	0,669495359
fieldset.html	0,652024413	0,649922322	0,630339891	0,684989844	0,645095950	0,658498651	0,655810276	0,690907932	0,828841676	0,723122629	0,681955358
figure-figcaption.html	0,655823102	0,666934471	0,680885690	0,898234682	0,634619583	0,629049775	0,631608102	0,641973809	0,684875797	0,733825617	0,685783063
footer.html	0,653280016	0,632666716	0,653337373	0,631092816	0,643769135	0,678412478	0,632207227	0,823758068	0,638188853	0,754652156	0,674136484
form.html	0,655268643	0,688410714	0,685456907	0,666686278	0,636444873	0,677043723	0,663067804	0,665708583	0,667010642	0,776360652	0,678145882
head.html	1,009058888	0,629744496	0,639692107	0,637240904	0,636102651	0,638046879	0,644857833	0,665354186	0,648952107	0,635711875	0,678476193
header.html	0,738088647	0,694029757	0,868349090	0,644987967	0,894264983	0,778838279	0,663560156	0,650492744	0,684080842	0,649892264	0,726658473
headings.html	0,669635238	0,638905895	0,661557802	0,661050175	0,647224749	0,654774647	0,855543475	0,688199506	0,677544914	0,657047542	0,681148394
hr.html	0,643902066	0,657648988	0,662696798	0,675551147	0,665999178	0,667204918	0,652697046	0,649218291	0,652533572	0,701552250	0,662900425
iframe.html	2,415552910	1,785110018	1,788254572	1,590061046	1,843921491	3,690546731	2,990322248	1,921861825	2,676349328	2,795496150	2,349747632
image.html	0,741084703	0,767595117	0,771085348	1,806737721	1,333183943	1,574940722	0,869593598	0,937081121	0,866137181	0,755550741	1,042299020
input.html	0,667840239	0,684310723	0,661477958	0,657324877	0,654447927	0,697533190	0,678151913	0,706678960	0,663505239	0,719911773	0,679118280
ins.html	0,750332613	0,808270538	0,649897070	0,634708740	0,648600274	0,670430602	0,650394872	0,653539666	0,638152364	0,650006905	0,675433364
italic.html	0,700477290	0,680420991	0,727077074	0,645742380	0,832494263	0,654748850	0,637176330	0,705812675	0,650046921	0,646901559	0,688089833
keygen.html	0,665632664	0,669858778	0,652408690	0,699502616	0,657729656	0,671549647	0,670437644	0,657869794	0,851310052	0,669401312	0,686570085
label.html	0,675029497	0,656390081	0,659920332	0,678624713	0,663641566	0,692877903	0,772253030	0,661133807	0,658884590	0,653237690	0,677199321
legend.html	0,637921192	0,663439492	0,807827289	0,643416174	0,642610912	0,650602177	0,639306456	0,653045713	0,674348714	0,641251307	0,665376943
li-ol-ul.html	0,967089810	0,782039635	0,793848208	0,775532530	1,335066611	0,803341950	0,793915450	0,853106392	0,886011853	0,797631619	0,878758406

link.html	0,672130123	0,645590380	0,663310054	0,659682958	0,647405311	0,676841845	0,674061872	0,817454289	0,655742913	0,646444938	0,675866468
lists.html	0,642971467	0,648942297	0,667818356	0,645043129	0,645219863	0,641191943	0,666275955	0,650769638	0,653260598	0,647053399	0,650854665
main.html	0,629716540	0,650543258	0,718627013	0,699454530	0,645630899	0,648805424	0,641773264	0,640472657	0,646011388	0,641310466	0,656234544
map.html	0,644040547	0,653884164	0,676286275	0,660947831	0,675755536	0,627968911	0,638164844	0,784236268	0,647511597	0,652058352	0,666085433
mark.html	0,639183593	0,661435176	0,656537514	0,643230790	0,720661906	0,638663468	0,640495203	0,648493321	0,649979208	0,648254035	0,654693421
menu-menuitem.html	0,645637370	0,634113809	0,728598876	0,811032258	0,643258403	0,643068559	0,638017043	0,636641479	0,645972175	0,657313371	0,668365334
meta.html	0,660136036	0,646528104	0,656084395	0,643524818	0,655292564	0,649029856	0,635661026	0,645658150	0,649574209	0,640276243	0,648176540
meter.html	0,655617418	0,673018132	0,661943013	0,651130201	0,660748195	0,670825896	0,666005878	0,656740775	0,640517731	0,645539854	0,658208709
nav.html	0,664575700	0,656171886	0,656808571	0,668198998	0,644532453	0,648619152	0,646742613	0,641748187	0,684203881	0,669626522	0,658122796
noscript.html	0,661192990	0,657657532	0,639335257	0,961274415	0,741672663	0,643727784	0,643610472	0,837454643	0,730337257	0,694843244	0,721110626
object.html	0,692491662	0,636757618	0,690662013	0,670226020	0,737277403	0,667691548	0,644748764	0,732870782	0,647179932	0,662076109	0,678198185
optgroup-option.html	0,660861150	0,695208354	0,649777513	0,684433754	0,651229353	1,082590008	0,805799301	0,868376188	0,660159512	0,663865903	0,742230104
output.html	0,653425976	0,656279096	0,648460792	0,676496235	0,851791800	0,673993014	0,642498185	0,647199051	0,646324005	0,661047623	0,675751578
paragraph.html	0,654981421	0,647809311	0,660426025	0,949033195	0,798449130	0,640924330	0,660848125	0,650641492	0,643400373	0,648219465	0,695473287
param.html	0,674908381	0,679944935	0,652227474	0,655545510	0,842296958	0,656753829	0,677779788	0,671940284	0,658442811	0,651899311	0,682173928
pre.html	0,676502081	0,635244929	0,643646691	0,636503691	0,650084543	0,820315013	0,815782115	0,640153544	0,669215260	0,644121087	0,683156895
progress.html	0,662324588	0,656657800	0,647158894	0,645010936	0,677961039	0,666326158	0,650980244	0,656794763	0,638869798	0,838335753	0,674041997
q.html	0,854605712	0,702847081	0,659434473	0,658096099	0,640497038	0,659915985	0,641281412	0,659143548	0,639992236	0,652362827	0,676817641
rp-rt-ruby.html	0,667386303	0,666600301	0,655152359	0,675773766	0,664842241	0,724540315	0,651852689	0,656521913	0,651664426	0,699646483	0,671398080
s(scratch).html	0,651747694	0,662327413	0,655174936	0,650987787	0,637139305	0,653938336	0,682525056	0,645412572	0,651509168	0,629905406	0,652066767
script.html	0,821110323	0,652451374	0,641944177	0,693139295	0,670980371	0,655757170	0,665306233	0,632317181	0,700527164	0,652830161	0,678636345
section.html	0,682727801	0,648391232	0,652099200	0,658075582	0,690826897	0,648049637	0,674521322	0,681104177	0,685961788	0,654726009	0,667648365
select.html	0,657186905	0,649922776	0,648400593	0,656747801	0,665821816	0,642900119	0,660753617	0,647463813	0,664728288	0,838368160	0,673229389
small.html	0,661759339	0,659877058	0,647697748	0,656616529	0,638563287	0,718537417	0,662339392	0,670248307	0,650441907	0,654600369	0,662068135
source.html	0,660963317	0,640982467	0,655787677	0,636832390	0,822065229	0,632568592	0,692832124	0,665649266	0,659907011	0,677931809	0,674551988
span.html	0,656643998	1,031558049	0,702836711	0,772820031	0,669099063	0,639080712	0,649805746	0,883663192	0,758794762	0,657518101	0,742182037
style.html	0,643007689	0,646936984	0,653134452	0,644883328	0,667761709	0,638753526	0,688622328	0,653457214	0,653061654	0,661658797	0,655127768
sub-sup.html	0,636233872	0,645385555	0,696280262	0,848331996	0,659127677	0,634491376	0,640681836	0,709285602	0,674702884	1,302763937	0,744728500
summary.html	0,895739736	1,104800452	1,441961887	1,067372686	0,650652941	0,907423438	0,671778784	0,647087752	0,652468113	0,672865416	0,871215120
tbody-thead-tfoot.html	0,641279622	0,663641695	0,663682818	0,650717365	0,638119915	0,655765870	0,921010038	0,838830195	0,639188139	0,636943794	0,694917945
textarea.html	0,770112696	0,646662679	0,666974797	0,659144901	0,638013249	0,636886063	0,646809789	0,647106545	0,628509472	0,693821478	0,663404167
time.html	0,639032203	0,653832029	0,646193474	0,640326411	0,654807155	0,790245130	0,787275760	0,644568772	0,657203069	0,648589930	0,676207393
title.html	0,654580627	0,650782415	0,670673547	0,656952494	0,673584548	0,654270201	0,677024342	0,707353278	0,653975592	0,644896732	0,664409378
track(is dynamic).html	0,639869938	0,811038820	0,663455878	0,680000862	0,641738379	0,674310833	0,646013112	0,676679997	0,654186001	0,643882784	0,673117660
underline.html	0,666109411	0,659304195	0,646041149	0,667108902	0,660567934	0,662272705	0,691016464	0,662593345	0,686091119	0,663646860	0,666475208
video.html	0,651355364	0,659970066	0,612710044	0,653195277	0,686601941	0,827231671	0,653485484	0,640756012	0,647137104	0,632667798	0,666511076
wbr.html	0,653343768	0,660204379	0,645824775	0,814396577	0,738823305	0,648552006	0,645677656	0,661129571	0,668608006	0,647685400	0,678424544

Appendix B

Chrome execution times

File	iteration 1	iteration 2	iteration 3	iteration 4	iteration 5	iteration 6	iteration 7	iteration 8	iteration 9	iteration 10	Average
a.html	2,687289406	2,046645332	2,012124876	2,040456804	2,113499101	2,104325376	2,077703514	2,071139515	2,075119518	2,032832063	2,126113551
abbreviation.html	2,037014058	2,862267398	2,422380026	2,176683456	2,107866939	2,191284573	2,160767238	2,146879036	2,230423541	2,228880953	2,256444722
address.html	2,090618129	2,132958125	2,166012137	2,146338800	2,076395601	2,163244561	2,192373661	2,179795826	2,133608502	2,150113877	2,143145922
area.html	2,161237695	2,177581370	2,151853865	2,144114867	2,119127916	2,181961708	2,159420270	2,187943097	2,150665820	2,134762374	2,156866898
article.html	2,195252240	2,080653719	2,110741053	2,159864488	2,234973436	2,194122112	2,151777735	2,163776281	2,168374085	2,243309060	2,170284421
aside.html	2,151225127	2,203773163	2,193812314	2,109739236	2,154575740	2,161901918	2,126246056	2,224371991	2,242151668	2,158474128	2,172627134
audio.html	2,258907481	2,207044721	2,204649144	2,177732096	2,186962208	2,232248809	2,147813256	2,183903397	2,103971435	2,269394351	2,197262690
base.html	2,108021877	2,194895439	2,195461712	2,125224130	2,113563189	2,135454444	2,807977690	2,268969013	2,073566087	2,078579143	2,210171272
bdi.html	2,057368598	2,058111162	1,977839788	2,010798468	2,134523540	2,078203947	1,998367858	2,041358441	2,137963733	2,033135031	2,052767057
bdo.html	2,104601692	2,086085512	1,996290753	2,083540558	2,043649643	2,057850850	2,016493944	2,065395852	2,004588831	2,054072541	2,051257018
blockquote.html	2,077729311	2,044137212	2,109262777	2,026033442	2,048713675	2,064064580	2,062769518	2,032855802	2,093912573	2,006242475	2,056572137
body.html	2,084766764	2,097055149	2,066768788	2,044035704	2,048402213	2,022652732	2,066276254	2,033740704	2,780367048	2,075744852	2,131981021
bold.html	2,066106903	2,002741703	2,035549879	2,054378463	2,034179810	2,071706485	2,099160933	2,081470616	2,032463285	2,068369880	2,054612796
breakline.html	2,096712676	2,059658650	2,096484212	2,048061565	2,121467472	2,139298425	2,066482308	2,110191142	2,141703718	2,025309998	2,090537017
button.html	2,152803226	2,038115923	2,060878143	2,023771641	2,044915789	2,142528511	2,020826483	2,323383811	2,516593290	2,687881678	2,201169850
canvas.html	2,348279669	2,530827399	2,875740592	2,484338251	2,070644405	2,042411590	2,983969275	2,098468680	2,037173660	2,061406110	2,353325963
caption.html	2,248293089	2,068341305	2,021468529	2,112037235	2,114740371	2,485935756	2,022600811	2,099806521	2,066538047	2,029762093	2,126952376
cite.html	2,173453206	2,064673935	1,961295687	2,079827514	2,063540645	2,080820677	2,121909643	2,084499055	2,273203997	2,083935048	2,098715941
code-em-strong-samp-var-kbd.html	2,108454545	2,104325280	2,112369049	2,086346983	2,038961685	2,106607705	2,109479223	2,106019100	2,099117772	2,063717718	2,093539906
col-table-tr-th-td.html	2,026004206	2,228791731	2,025603397	2,102865033	2,103924470	2,353614191	2,046175648	2,033697153	2,038814872	2,028437455	2,098792816
colgroup.html	2,128708935	2,018514333	2,074286570	2,116548507	1,990975442	2,062460183	2,054864874	2,171561877	2,009091303	2,043814507	2,067082653
comments.html	1,998353398	2,042811869	2,002793745	2,072917066	2,094640476	2,256263552	2,044960618	1,994630321	2,060815565	2,071390519	2,063957713
datalist.html	2,054995999	2,159144788	2,667304848	2,042312223	2,105736924	2,025474264	2,111988937	2,096973138	2,062404355	2,037549479	2,136388495
dd.html	1,991027587	2,052646972	2,072385814	2,065431726	2,036248589	2,136233216	2,035220794	2,053283599	2,056409469	2,037611860	2,053649963
del.html	2,031529890	2,032691209	2,048326778	2,084598824	2,059954128	2,079619952	2,121830481	2,090638528	2,375546412	2,157285775	2,108202198
details.html	2,112404220	2,065614442	2,042907035	2,170369002	2,018803615	2,058983443	2,046493367	2,119526114	2,075697546	2,167473902	2,087827269
dfn.html	2,052340672	2,049116738	2,203006628	2,060538701	2,133662184	2,060715837	2,024466064	2,112819204	2,044915893	2,037136340	2,077871826
dialog.html	2,061104048	2,053296356	2,045376124	2,101836183	2,094131522	2,044938650	2,081579017	2,001331733	2,023010231	2,000161032	2,050676490
div.html	2,148440959	2,053745935	2,035102489	2,043540404	2,071308731	2,097146952	2,017398401	2,134700385	1,983880548	2,068212209	2,065347701
dl-dt-dd.html	2,014632714	2,058370342	2,049409973	2,113017883	1,990695737	2,044283719	2,056940166	2,057750702	2,064526879	2,130006066	2,057963418
embed.html	2,513607641	2,607314366	2,468721447	2,441071293	2,385563393	2,502327333	2,511228498	2,457818290	2,501736080	2,428573386	2,481796173
fieldset.html	2,049501285	2,002440965	2,039273853	2,266542625	2,098852582	2,214045322	2,206996914	2,252439653	2,184900347	2,267268163	2,158226171
figure-figcaption.html	2,273136320	2,178426637	2,071316444	2,184382404	2,249043192	2,104131926	2,189511291	2,132268078	2,197234294	2,193612131	2,177306272
footer.html	2,087021551	2,242603340	2,120553048	2,225519071	2,149607795	2,287456806	2,054910646	1,984543897	2,090263221	2,024205208	2,126668458
form.html	2,024190004	2,061531760	2,095615182	2,036597728	2,103928941	2,059959467	2,020504048	2,046055258	2,063768732	1,995892871	2,050804399
head.html	2,046101146	2,064368805	2,120986183	2,048696326	2,070484904	2,063949149	2,043294329	2,010623634	2,066362765	2,061520583	2,059638782
header.html	2,091344068	2,030656514	2,134330940	2,036153798	2,026293295	2,081946081	2,092045339	2,092736462	2,093972351	2,245472356	2,092495120
headings.html	2,074688613	2,249491179	2,064920896	2,273446052	2,121975493	2,141306656	2,320480991	2,139780267	2,049761348	2,166302671	2,160215417
hr.html	2,082457020	2,185472988	2,137024651	2,048129399	2,114364775	2,074946086	2,271431724	2,061704809	1,996997080	2,039028319	2,101155685
iframe.html	3,333283320	3,303698273	3,384316347	3,374063045	3,363176772	3,149759796	3,143226213	3,212215555	3,510330023	3,225823670	3,299989301
image.html	2,066121315	2,388004006	2,160187569	2,076830305	2,065891477	2,151105202	2,065960315	2,049797459	2,048931513	2,062661072	2,113549023
input.html	2,150032928	2,039298608	2,071308666	2,075681153	2,017549325	2,067489292	2,066205535	2,065883314	2,047568124	2,024225241	2,062524219
ins.html	2,079424857	2,071816707	2,095386918	2,115396700	2,085184272	2,043137982	2,063535125	2,118171707	2,029394716	2,066307195	2,076775618
italic.html	2,011303499	2,038774388	2,194085107	2,082634563	2,127521559	2,045722952	2,045859531	1,992827539	2,151871829	2,114395717	2,080499668
keygen.html	2,121517721	2,105337750	2,060309996	2,091438738	2,118128796	2,116892538	2,305207578	2,119453751	2,125769513	2,326701129	2,149075751
label.html	2,243947298	2,136861487	1,938108519	2,169501008	2,216124004	2,163495164	2,062284469	2,028947223	2,054350500	2,012333326	2,102595300
legend.html	2,014583290	2,075763128	2,109964221	2,124547976	2,037228978	2,310748231	2,039862151	2,058479521	2,545227812	2,055670932	2,137207624
li-ol-ul.html	3,355636906	2,444677048	2,491701511	2,407999305	3,013323059	2,889519860	2,850658205	3,451144148	3,124676176	4,644536234	3,067387245

link.html	3,378559599	2,322745945	2,765802927	2,256895853	2,275587122	2,097696310	2,007377040	2,145334895	2,111996038	2,075660941	2,343765667
lists.html	2,061857240	2,072939285	2,072954046	2,047470107	1,978587344	2,051689913	2,086182696	2,037835466	2,105571526	2,086555327	2,060164295
main.html	2,160985207	2,167632600	2,099135552	2,019725359	1,984863086	2,069018537	2,015060310	2,034042524	1,974808504	2,091856076	2,061712775
map.html	2,050537956	2,015659053	2,030863222	2,170471482	2,070181947	2,126704580	2,049519220	2,115918244	2,106793113	2,058366335	2,079501515
mark.html	2,052736979	2,104198757	2,075846667	2,104058981	2,053431244	2,038144164	2,090640520	2,052689580	2,131805407	2,202317166	2,090586947
menu-menuitem.html	2,090388776	2,100661682	2,017356709	2,080378322	2,115795674	2,028379876	2,080798099	2,051755896	2,110990234	2,072055130	2,074856040
meta.html	2,167258235	2,184428372	2,030429603	2,146203995	2,091034057	2,049148685	2,085659730	2,025049329	2,115319577	2,083966195	2,097849778
meter.html	2,062858565	2,136887031	2,064197704	2,150133801	2,186798865	2,004318209	2,102260352	2,066205662	2,095184722	2,058194068	2,092703898
nav.html	2,042327259	2,140916016	2,043352649	2,155430680	2,058656905	2,061205095	2,089920252	2,086772894	2,078348119	2,045167543	2,080209741
noscript.html	2,092700325	2,044223035	2,058259543	2,051898549	2,038341819	2,066473705	2,060080736	2,077393873	2,071216592	2,058430497	2,061901867
object.html	2,527108907	2,467040348	2,666710819	2,591066064	2,432781619	2,466163782	2,435585968	2,512982224	2,517408584	2,537075801	2,515392412
optgroup-option.html	2,030401026	2,066543206	2,038367489	2,123947973	2,051738038	2,324653456	2,139531266	2,060142991	2,100831867	2,074857213	2,101101453
output.html	2,084390065	2,166115332	2,038421235	1,980532192	2,182466858	2,107718293	2,040966563	2,194287583	2,111721526	2,073050309	2,097966996
paragraph.html	2,209858466	2,041895493	1,981742095	2,200162617	2,082995262	2,123476994	2,072854787	2,079746554	2,145805979	2,208010889	2,114654914
param.html	2,657713432	2,458750962	2,334995906	2,353231946	2,285784989	2,228090015	2,784158575	4,613050164	2,605977589	2,822641122	2,714439470
pre.html	2,257062141	2,307290434	2,164319580	2,394426801	2,238695816	2,239985545	2,200751755	2,268489647	2,139376902	2,206933719	2,241733234
progress.html	2,791958034	2,570262097	3,445068210	3,165025671	2,273009186	2,138286374	2,113619399	2,132140398	2,121325083	2,082278050	2,483297250
q.html	2,129185571	2,114296497	2,196861454	2,071826688	2,108315753	2,136366959	2,131750486	2,210467855	2,071122939	2,130359842	2,130055404
rp-rt-ruby.html	2,084485160	2,092125398	2,074525128	2,072980503	2,108855014	2,066774736	2,120032809	2,112439577	2,133430768	2,098998576	2,096464767
s(scratch).html	2,086231426	2,243901301	2,094366747	2,077559933	2,108381392	2,090568471	2,086706684	2,138531075	2,380077283	2,141984017	2,144830833
script.html	2,082351354	2,105411404	2,111593230	2,041546611	2,134389876	2,225478414	2,030616676	2,081649085	2,111652762	2,093140007	2,101782942
section.html	2,105862208	2,108245806	2,085446084	2,140138552	2,104650264	2,114508958	2,078606475	2,159663551	2,569699647	2,550825161	2,201764671
select.html	3,310810518	2,228663704	2,276630988	2,228470769	2,099445241	2,158012833	2,114707918	2,109093624	2,172021609	2,127204765	2,282506197
small.html	2,306142617	2,231815550	2,213977116	2,164196964	2,146690147	2,152461143	2,125385094	2,159623708	2,115740763	2,144552312	2,176058541
source.html	2,646699917	2,276532250	2,212881339	2,264693999	2,360620173	2,277898317	2,301259524	2,278159158	2,290062702	2,285586846	2,319439423
span.html	2,189310629	2,094151106	2,115654028	2,085681283	2,091464299	2,145072397	2,157943508	2,192139073	2,133477091	2,177389287	2,138228270
style.html	2,059324286	2,135472502	2,041039271	2,100527253	2,103928592	2,122239368	2,083038016	2,125934521	2,145468965	2,105493577	2,102246635
sub-sup.html	2,162046409	2,190777416	2,281574934	2,122644605	2,253230386	2,315586211	2,475105545	2,043216719	2,036388417	2,093022243	2,197359289
summary.html	2,095009884	2,062127038	2,075041704	2,127100872	2,093949003	2,161369109	2,083880277	2,096404692	2,105327628	2,224225986	2,112443619
tbody-thead-tfoot.html	2,195668750	2,160282197	2,135015022	2,165180782	2,198978449	2,110199280	2,176459083	2,160173739	2,244211324	2,048350234	2,159451886
textarea.html	2,121667163	2,194631235	2,183630484	2,139758242	2,075761501	2,096376922	2,197461421	2,095338474	2,158061108	2,125526575	2,138821312
time.html	2,241597383	2,233078472	2,122458159	2,187693079	2,144846616	2,166237026	2,232837207	2,416398121	2,110275043	2,069745087	2,192516619
title.html	2,048726575	2,096094551	2,083916552	2,112959364	2,098498775	2,172271792	2,129905720	2,032782269	2,111785620	2,096449009	2,098339023
track(is dynamic).html	3,035444843	2,917889610	2,650953786	2,561985534	2,667517301	3,339960559	2,679252546	2,547520781	2,574260078	2,550891216	2,752567625
underline.html	2,120372183	2,081440986	2,044950091	2,116241075	2,085896344	2,016059218	2,040138020	2,131066687	2,097392866	2,079882259	2,081343973
video.html	2,837083528	2,538016484	2,438594372	2,429275695	2,388749576	3,910371971	3,144978588	3,069188584	2,848466830	3,591306336	2,919603196
wbr.html	2,123389160	2,241256229	2,104211337	2,089898367	2,097222479	2,097832249	2,156471137	2,111538730	2,080642189	2,100919410	2,120338129

Chrome execution times

Appendix C

Firefox execution times

file	iteration 1	iteration 2	iteration 3	iteration 4	iteration 5	iteration 6	iteration 7	iteration 8	iteration 9	iteration 10	average
a.html	4,426469989	3,443261298	3,430124446	3,402664042	3,412692514	3,370882332	3,378045725	3,391386749	3,342336401	3,367867929	3,496573143
abbreviation.html	3,414371829	3,425932960	3,355460370	3,433025941	3,353561385	3,394043895	3,416432381	3,445467012	3,460570654	3,354669629	3,405353606
address.html	3,377132483	3,296287675	3,364907195	3,327044308	3,444437497	3,339462859	3,489413773	3,408814995	3,342950054	3,375201325	3,376565216
article.html	3,348861742	3,345239944	3,408853308	3,413812471	3,407214656	3,588802167	3,526947809	3,360493176	3,366309690	3,339155419	3,410569038
area.html	3,367669216	3,329158726	3,356862203	3,356578853	3,470970293	3,429734862	3,427086330	3,583568987	3,417071924	3,382344534	3,412104593
aside.html	3,350484827	3,353675546	3,319140063	3,440919804	3,368257904	3,427867607	3,430154611	3,482244754	3,446679887	3,541981819	3,416140682
audio.html	3,415783618	3,417831957	3,384398203	3,366687917	3,396526036	3,419375935	3,395683817	3,434357490	3,389199706	3,362098341	3,398194302
base.html	3,590563090	3,397419660	3,418767236	3,502535223	3,517881137	3,477969585	3,517131761	3,480610598	3,643955272	3,432831950	3,497966551
bdi.html	3,314126036	3,302026060	3,357289931	3,286244687	3,349177502	3,517226378	3,308319796	3,469575627	3,338633570	3,371250509	3,361387010
bdo.html	3,365591138	3,362539682	3,320161198	3,307998656	3,412178286	3,406573859	3,400982141	3,358560195	3,408939294	3,490172095	3,383369654
blockquote.html	3,470034343	3,531268911	3,434626811	3,622758548	3,350157565	3,415591834	3,368332424	3,411607771	3,320309836	3,341695943	3,426638399
body.html	3,318610513	3,317106592	3,402372903	3,334182159	3,446504494	3,384588723	3,393821246	3,370935374	3,465296607	3,340292318	3,377371093
breakline.html	3,642288661	3,366196499	3,322009024	3,418471371	3,384011176	3,458769992	3,466507279	3,361407788	3,476675845	3,483841604	3,438017924
bold.html	3,505846502	3,718238654	3,626469161	3,413508515	3,460248449	3,371370508	3,438941011	3,391053130	3,363253280	3,330284135	3,461921335
button.html	3,342858818	3,421068875	3,460709783	3,385758679	3,405982003	3,372271590	3,426852708	3,337018585	3,392564688	3,378405809	3,392349154
canvas.html	3,518328369	3,464979998	3,457341334	3,595832564	3,561445940	3,478267559	3,594045319	3,670490588	3,604974707	3,610667242	3,555637362
caption.html	3,730303192	3,836500782	3,574110287	3,710782076	3,543466597	3,533788246	3,588060764	3,443837329	3,470164849	3,485238418	3,591625254
cite.html	3,556847248	3,549942835	3,501992217	3,462641296	3,510261663	3,433818303	3,520462338	3,582056972	3,417120901	3,530911934	3,506605571
code-em-strong-samp-var-kbd.html	3,463431223	3,524788319	3,459417749	3,493296753	3,517689102	3,629304080	3,532854239	3,580809647	3,510830653	3,570095267	3,528251703
col-table-tr-th-td.html	3,636133029	3,559653040	3,730665129	3,829847204	3,836899319	3,863651946	3,889741007	3,357441601	3,393857678	3,512919378	3,661080933
colgroup.html	3,423889401	3,372973724	3,700024890	3,483423863	3,396980869	3,499281277	3,357539659	3,366513289	3,519899436	3,344343868	3,446487028
comments.html	3,389403877	3,488945113	3,351515942	3,437837901	3,452471813	3,341191214	3,354351358	3,489550835	3,304522628	3,368444027	3,397823471
datalist.html	3,578508967	3,350645941	3,387483103	3,474801662	3,559958136	3,434470420	3,501760019	3,636000211	3,595147644	3,720670293	3,523944640
dd.html	3,615495507	3,582405864	3,807311107	3,899471236	3,907693998	3,451109453	3,436407580	3,518222344	3,398126258	3,406777334	3,602302068
del.html	3,553968450	3,406920273	3,417659753	3,373792478	3,455139329	3,597524204	3,435713599	3,418974709	3,512570440	3,422097669	3,459436090
details.html	3,464397629	3,475291736	3,424549242	3,490116940	3,562271455	3,558984562	3,518622417	3,703150359	3,650106776	3,448469702	3,529596082
dfn.html	3,509966570	3,515855510	3,460045964	3,487476904	3,881595507	3,941683572	4,095172844	4,272385820	4,414342753	4,488602239	3,906712768
dialog.html	4,404017247	4,304976232	4,430364226	3,484417184	3,545776363	3,403150847	3,562465986	3,633541294	3,501421979	3,608007472	3,787813883
div.html	3,750061160	4,059334272	3,745088034	3,700159479	3,624968537	3,449765882	3,561946174	3,636511701	3,643788342	3,618670944	3,679029453
dl-dt-dd.html	4,127348785	3,377548372	3,551599785	3,572725874	3,280091383	3,369776812	3,491528630	3,356839982	3,410519050	3,501323832	3,503930251
fieldset.html	3,566277904	3,327389808	3,321684725	3,459709912	3,517556532	3,551595462	3,355520542	3,463428882	3,620561584	3,388278680	3,457200403
embed.html	3,349006846	3,462827295	3,278062081	3,340008372	3,321898241	3,443835145	3,345035439	3,356264068	3,416846062	3,566994542	3,388077809
figure-figcaption.html	4,023045211	4,155492119	4,358358162	4,388101989	4,457176934	4,578110959	4,583445546	5,135711318	5,047125660	5,148968583	4,587553648
footer.html	3,956805803	3,997515580	3,863528650	4,393845553	3,885761142	3,857103413	3,936728379	4,016785462	4,010479499	4,031195668	3,994974915
form.html	4,166659333	4,348255470	4,068927857	4,253069082	4,136486232	4,514869291	4,070128720	4,361912129	4,283449261	4,771468003	4,297522538
head.html	3,982796506	3,855931506	3,899986164	3,783679502	3,978569767	4,302088462	3,934870016	4,108895839	3,727438457	4,081541841	3,965579806
header.html	3,845820729	3,896692598	3,930655626	3,772846535	4,159458559	4,014598594	3,863639607	4,041219438	3,877067533	4,070778155	3,947277737
headings.html	3,980281635	4,016428874	4,292120344	3,934204423	4,027093446	4,133713135	4,915758461	4,833083933	5,701036728	5,338671984	4,517239296
hr.html	5,019580112	5,358652815	4,937903768	5,125090461	4,220622558	3,953195896	3,943456529	3,873592336	3,921868426	4,086816799	4,444077970
iframe.html	4,930119530	5,028855905	6,016997977	5,011204477	4,621853282	4,598028619	4,763478700	4,860856811	4,658978268	4,642003520	4,913237709
image.html	3,733070926	3,899076021	3,669327334	3,677002374	3,965913120	3,479050786	3,430057711	3,549242448	3,655854973	3,470665112	3,652926081
input.html	3,343164974	3,483231596	3,550645035	3,358358999	3,396166863	3,365074419	3,454507648	3,627904388	3,378751752	3,389501090	3,434730676
ins.html	3,363509406	3,440149150	3,523317035	3,382834342	3,531751805	3,423396953	3,373317273	3,567201802	3,569692551	3,545528547	3,472069886
italic.html	3,359719702	3,336947668	3,472129892	3,716402668	3,419908689	3,453593022	3,503528999	3,569809917	3,693749040	3,796937618	3,532272722
keygen.html	3,553781134	4,008234921	3,750743375	3,845613245	3,983715890	3,948891516	3,985701733	3,545313582	3,385424936	3,426597472	3,743401780
label.html	3,524503666	3,470682067	3,630211684	3,634308524	4,054174370	3,487984856	3,573775814	3,483656531	3,649124229	3,349311784	3,585773353
legend.html	3,561462316	3,453837402	3,401229183	3,634117126	3,585844905	3,415990863	3,661551568	3,497457424	3,532826509	4,905128703	3,664944600
li-ol-ul.html	4,876500042	3,992917223	4,258225372	4,693564818	4,584514925	5,424256782	4,123810046	3,933708375	4,013171957	4,241379288	4,414204883

link.html	6,441118925	5,158974195	3,424888950	3,549378680	3,560705220	4,150611083	4,151038181	4,385726307	4,236175272	4,346460516	4,340507733
lists.html	4,447070287	4,416875029	4,499075194	4,119059498	4,568954149	3,507183256	3,399485376	3,328999128	3,497929033	3,405644783	3,919027573
main.html	3,408043772	3,430988584	3,580322339	3,675161731	3,403877327	3,638278650	3,402922554	3,542504498	3,566721867	3,723567020	3,537238834
map.html	3,787863124	3,524177854	3,592045059	3,636443511	3,564765143	3,614833380	3,829615348	3,392087064	3,366487013	3,399031223	3,570734872
mark.html	3,609592061	3,524136942	3,320669139	3,561610842	3,364250568	3,372826504	3,366158027	3,449631162	3,630865338	3,580984631	3,478072521
menu-menuitem.html	3,384008603	3,585642476	3,431644133	3,328304696	3,354905339	3,433435653	3,604706509	3,597801034	3,344702923	3,549847803	3,461499917
meta.html	3,383296973	3,397258041	3,355515966	3,428842428	3,561822187	3,603845816	3,542135853	3,714217631	3,959032793	3,907853304	3,585382099
meter.html	3,930149957	4,065106609	4,143641733	4,196808449	4,335731339	4,209134372	4,084404467	4,273854259	4,376684571	4,218502357	4,183401811
nav.html	4,217772752	3,684861440	3,575994877	3,579399708	3,662857239	3,747437000	3,551621926	3,678815187	3,579925888	3,813082816	3,709176883
noscript.html	3,672211262	3,520347812	3,601790614	3,785204877	3,819964119	3,802248163	3,564692633	3,648971545	3,917743588	3,705583399	3,703875801
object.html	3,876279323	3,719633870	4,005151077	3,528666247	3,545543951	3,779666670	3,684371137	3,578412673	3,521585829	3,779591487	3,701890226
optgroup-option.html	3,705443335	3,735745993	3,523755876	3,703739442	3,585877987	3,523447690	3,512658199	3,682720572	3,824380712	3,769574466	3,656734427
output.html	3,859675078	3,522465611	3,588529036	3,525583813	3,541457774	3,602795163	3,701906056	3,742158775	3,746982579	3,991900905	3,682345479
paragraph.html	4,194054428	4,112782967	4,108455238	4,288683149	4,344825962	4,382691791	4,323162985	4,364571963	4,283673542	4,437530373	4,284043240
param.html	4,598369438	4,604546974	3,701564687	3,625331602	3,657161942	3,727116336	3,802091708	3,612131608	3,757018123	3,742006916	3,882733933
pre.html	3,594008889	3,546818060	3,551545480	3,705212153	3,858231222	3,912569638	3,879264490	3,870749288	3,799798380	3,702849098	3,742104670
progress.html	4,020489806	3,554216185	3,529178705	3,591768599	3,497874935	3,589401457	3,732484393	3,715810516	3,718433388	3,703142595	3,665280058
q.html	3,488962027	3,525389308	3,525451939	3,541808497	3,528259649	3,524126760	3,456889471	3,578674691	3,726270769	3,721429986	3,561726310
rp-rt-ruby.html	3,682634060	3,731096211	3,554923381	3,491745773	3,881541735	3,569911065	3,790442398	3,641025270	3,564676750	3,604218998	3,651221564
s(scratch).html	3,745766139	3,825620237	3,800335016	3,830103562	3,934315828	3,964478369	4,188895581	3,480455734	3,487167748	3,528136149	3,778527436
script.html	3,480810476	3,468117532	3,550565201	3,735430671	3,638209975	3,660581699	3,716549933	3,666902208	3,718704268	3,517459578	3,615333154
section.html	3,539462314	3,683531665	3,569026671	3,595496218	3,493725439	3,519202013	3,523133658	3,499049733	3,548952994	3,693011359	3,566459206
select.html	3,663408759	3,660050262	3,687731632	3,692579438	3,684464453	3,899721560	3,988297700	4,047905351	4,044476852	3,992486035	3,836112204
small.html	4,172102926	4,234363782	4,171791423	4,712230618	4,295137673	4,441227252	4,640137698	3,783157376	3,665235030	3,672686685	4,178807046
source.html	3,732233040	3,831629017	3,816901865	3,754975376	3,773939387	3,726646240	3,783257259	3,810713702	3,837006414	3,858396459	3,792569876
span.html	3,764627109	3,862898269	4,009348384	3,786824710	3,729464307	3,632849319	3,560949885	3,612445948	3,629072627	3,719943120	3,730842368
style.html	3,562131369	3,665758990	3,821291231	3,677303328	3,873612743	3,663814264	3,627520262	3,653223208	3,629609153	3,636732097	3,681099664
sub-sup.html	3,587077378	3,592488057	3,584779721	3,696403553	3,642217653	3,718471947	3,810849480	3,970554140	3,917381893	3,897966681	3,741819050
summary.html	4,116542023	4,041780483	3,728317785	3,710929217	3,604213631	3,581563879	3,608395315	3,628665079	3,596095931	3,702479957	3,731898330
tbody-thead-tfoot.html	3,766490739	3,754613816	3,743466972	3,696630489	3,719218498	3,732289055	3,670054015	3,717657176	3,711013352	3,712460998	3,722389511
textarea.html	3,713723362	3,758950363	3,808434919	4,010602335	3,633010865	3,722785803	3,781485367	3,746500519	3,688453744	3,753284054	3,761723133
time.html	3,701373140	3,729809426	3,707663462	3,774506814	3,782800916	3,727372764	3,743641020	3,772725255	3,748139094	3,788247831	3,747627972
title.html	3,858198349	3,839772983	3,791981274	3,805266464	3,782179658	3,827675049	3,868857290	3,994195852	3,638647753	3,733569672	3,814034434
track(is dynamic).html	5,279452882	4,539125621	4,517094414	4,297040022	5,054658149	4,411060200	4,791916633	4,472945022	4,390283963	4,871273479	4,662485039
underline.html	3,582981134	3,661563237	3,650408303	3,681691510	3,789007419	3,818115717	3,917402792	3,643411544	3,645598820	3,627846845	3,701802732
video.html	4,499913418	4,228043275	4,282046346	4,067727493	4,130047182	4,230512137	4,063975848	4,168757735	4,078305121	4,038734312	4,178806287
wbr.html	3,652203008	3,594639215	3,611056942	3,729106129	3,730911903	3,812264503	3,647112028	3,628473034	3,651184095	3,755894142	3,681284500

Firefox execution times

Appendix D

Group test data

Basic HTML

	Phantom	Chrome	Firefox
Title	0,664409378	2,098339023	3,814034434
Body	0,670729346	2,131981021	3,377371093
H1 to H6	0,681148394	2,160215417	4,517239296
P	0,695473287	2,114654914	4,284043240
Br	0,673700368	2,090537017	3,461921335
Hr	0,662900425	2,101155685	4,444077970
Comment	0,633254801	2,063957713	3,397823471
Min	0,633254801	2,063957713	3,377371093
Q1	0,663654902	2,09443802	3,429872403
Median	0,670729346	2,101155685	3,814034434
Q3	0,677424381	2,123317967	4,364060605
Max	0,695473287	2,160215417	4,517239296
Box 1 (hidden)	0,663654902	2,09443802	3,429872403
Box 2 (lower)	0,007074444	0,006717665	0,384162032
Box 3 (upper)	0,006695035	0,022162282	0,550026171
Whisker Top	0,018048906	0,036897449	0,153178691
Whisker Bottom	0,030400100	0,030480307	0,052501310
Range	0,062218485	0,096257704	1,139868203
Factor of Range Differance between Phantom and Firefox			18,32041069
[25-75] percentile range	0,013769480	0,028879948	0,934188202

Formating HTML

	Phantom	Chrome	Firefox
Abbr	0,769462620	2,256444722	3,405353606
Address	0,786580185	2,143145922	3,376565216
B(bold)	0,672505768	2,054612796	3,438017924
Bdi	0,699961581	2,052767057	3,361387010
Bdo	0,701696079	2,051257018	3,383369654
Blockquote	0,666441396	2,056572137	3,426638399
Cite	0,673912493	2,098715941	3,506605571
Code-Em-Strong-Samp-Var-Kbd	0,656786909	2,093539906	3,528251703
Del	0,663051817	2,108202198	3,459436090
Dfn	0,678309018	2,077871826	3,906712768
I(Italic)	0,688089833	2,080499668	3,532272722
Ins	0,675433364	2,076775618	3,472069886
Mark	0,654693421	2,090586947	3,478072521
Meter	0,658208709	2,092703898	4,183401811
Pre	0,683156895	2,241733234	3,742104670
Progress	0,674041997	2,483297250	3,665280058
Q	0,676817641	2,130055404	3,561726310
Rp-rt-ruby	0,671398080	2,096464767	3,651221564
S(scratch)	0,652066767	2,144830833	3,778527436
Small	0,662068135	2,176058541	4,178807046
Sub-sup	0,744728500	2,197359289	3,741819050
Time	0,676207393	2,192516619	3,747627972
U(Underline)	0,666475208	2,081343973	3,701802732
Wbr	0,678424544	2,120338129	3,681284500
Min	0,652066767	2,051257018	3,361387010
Q1	0,665594001	2,079842708	3,454081549
Median	0,674737681	2,097590354	3,546999516
Q3	0,68439013	2,15263776	3,741890455
Max	0,786580185	2,483297250	4,183401811
Box 1 (hidden)	0,665594001	2,079842708	3,454081549
Box 2 (lower)	0,009143680	0,017747646	0,092917967
Box 3 (upper)	0,009652449	0,055047406	0,194890940
Whisker Top	0,102190055	0,330659490	0,441511356
Whisker Bottom	0,013527234	0,028585690	0,092694539
Range	0,134513418	0,432040233	0,822014802
Factor of Range Differance between Phantom and Firefox			6,111024574
[25-75] percentile range	0,018796129	0,072795052	0,287808906

Lists and Tables HTML

	Phantom	Chrome	Firefox
Li-Ol-Ul	0,878758406	3,067387245	4,414204883
Dl-Dt-Dd	0,655271065	2,057963418	3,503930251
Menu-Menuitem	0,668365334	2,074856040	3,461499917
Col-Table-Tr-Th-Td	0,668130821	2,098792816	3,661080933
Caption	0,685769960	2,126952376	3,591625254
Tbody-Thead-Tfoot	0,694917945	2,159451886	3,722389511
Colgroup	0,673256521	2,067082653	3,446487028
Min	0,655271065	2,057963418	3,446487028
Q1	0,668248078	2,070969346	3,482715084
Median	0,673256521	2,098792816	3,591625254
Q3	0,690343952	2,143202131	3,691735222
Max	0,878758406	3,067387245	4,414204883
Box 1 (hidden)	0,668248078	2,070969346	3,482715084
Box 2 (lower)	0,005008443	0,027823469	0,108910170
Box 3 (upper)	0,017087432	0,044409315	0,100109968
Whisker Top	0,188414453	0,924185114	0,722469661
Whisker Bottom	0,012977012	0,013005928	0,036228056
Range	0,223487340	1,009423827	0,967717855
Factor of Range Differance between Phantom and Firefox			4,330079071
[25-75] percentile range	0,022095875	0,072232784	0,209020138

Form and input HTML

	Phantom	Chrome	Firefox
Form	0,678145882	2,050804399	4,297522538
Input	0,679118280	2,062524219	3,434730676
Textarea	0,663404167	2,138821312	3,761723133
Button	0,668198322	2,201169850	3,392349154
Select	0,673229389	2,282506197	3,836112204
Optgroup-Option	0,742230104	2,101101453	3,656734427
Fieldset	0,681955358	2,158226171	3,388077809
Legend	0,665376943	2,137207624	3,664944600
Datalist	0,688409309	2,136388495	3,523944640
Keygen	0,686570085	2,149075751	3,743401780
Output	0,675751578	2,097966996	3,682345479
Min	0,663404167	2,050804399	3,388077809
Q1	0,670713855	2,099534224	3,479337658
Median	0,678145882	2,137207624	3,664944600
Q3	0,684262722	2,153650961	3,752562457
Max	0,742230104	2,282506197	4,297522538
Box 1 (hidden)	0,670713855	2,099534224	3,479337658
Box 2 (lower)	0,007432027	0,037673400	0,185606942
Box 3 (upper)	0,006116840	0,016443337	0,087617857
Whisker Top	0,057967382	0,128855236	0,544960081
Whisker Bottom	0,007309688	0,048729825	0,091259849
Range	0,078825937	0,231701798	0,909444729
Factor of Range Differance between Phantom and Firefox			11,53737928
[25-75] percentile range	0,013548867	0,054116737	0,273224799

Dynamic HTML

	Phantom	Chrome	Firefox
iFrame	2,349747632	3,299989301	4,913237709
Audio	0,796830538	2,197262690	3,398194302
Source	0,674551988	2,319439423	3,792569876
Track	0,673117660	2,752567625	4,662485039
Video	0,666511076	2,919603196	4,178806287
Min	0,666511076	2,197262690	3,398194302
Q1	0,67311766	2,319439423	3,792569876
Median	0,674551988	2,752567625	4,178806287
Q3	0,796830538	2,919603196	4,662485039
Max	2,349747632	3,299989301	4,913237709
Box 1 (hidden)	0,67311766	2,319439423	3,792569876
Box 2 (lower)	0,001434328	0,433128203	0,386236411
Box 3 (upper)	0,122278550	0,167035571	0,483678752
Whisker Top	1,552917094	0,380386105	0,250752670
Whisker Bottom	0,006606584	0,122176733	0,394375574
Range	1,683236556	1,102726612	1,515043407
Factor of Range Differance between Phantom and Firefox			0,900077533
[25-75] percentile range	0,123712878	0,600163774	0,869915163

Style and Semmantics HTML

	Phantom	Chrome	Firefox
Style	0,655127768	2,102246635	3,681099664
Span	0,742182037	2,138228270	3,730842368
Header	0,681148394	2,160215417	4,517239296
Footer	0,726658473	2,092495120	3,947277737
Main	0,656234544	2,061712775	3,537238834
Section	0,667648365	2,201764671	3,566459206
Article	0,835194708	2,170284421	3,412104593
Aside	0,742072496	2,172627134	3,416140682
Details	0,657280065	2,087827269	3,529596082
Dialog	0,670184107	2,050676490	3,787813883
Summary	0,871215120	2,112443619	3,731898330
Min	0,655127768	2,050676490	3,412104593
Q1	0,662464215	2,090161194	3,533417458
Median	0,681148394	2,112443619	3,681099664
Q3	0,742127266	2,165249919	3,759856106
Max	0,871215120	2,201764671	4,517239296
Box 1 (hidden)	0,662464215	2,090161194	3,533417458
Box 2 (lower)	0,018684179	0,022282425	0,147682206
Box 3 (upper)	0,060978872	0,052806299	0,078756442
Whisker Top	0,129087854	0,036514752	0,757383190
Whisker Bottom	0,007336447	0,039484705	0,121312865
Range	0,216087352	0,151088181	1,105134703
Factor of Range Differance between Phantom and Firefox			5,114296098
[25-75] percentile range	0,079663051	0,075088724	0,226438648

Links+Images HTML

	Phantom	Chrome	Firefox
Image	1,042299020	2,113549023	3,652926081
A	0,695960841	2,126113551	3,496573143
Nav	0,658122796	2,080209741	3,709176883
Min	0,658122796	2,080209741	3,496573143
Q1	0,677041819	2,096879382	3,574749612
Median	0,695960841	2,113549023	3,652926081
Q3	0,86912993	2,119831287	3,681051482
Max	1,042299020	2,126113551	3,709176883
Box 1 (hidden)	0,677041819	2,096879382	3,574749612
Box 2 (lower)	0,018919022	0,016669641	0,078176469
Box 3 (upper)	0,173169089	0,006282264	0,028125401
Whisker Top	0,173169089	0,006282264	0,028125401
Whisker Bottom	0,018919022	0,016669641	0,078176469
Range	0,384176223	0,045903809	0,212603741
Factor of Range Differance between Phantom and Firefox			0,553401611
[25-75] percentile range	0,192088112	0,022951905	0,106301870

Meta and Programming HTML

	Phantom	Chrome	Firefox
Head	0,678476193	2,059638782	3,965579806
Meta	0,648176540	2,097849778	3,585382099
Base	0,773343804	2,210171272	3,497966551
Script	0,678636345	2,101782942	3,615333154
NoScript	0,721110626	2,061901867	3,703875801
Embed	0,669495359	2,481796173	3,457200403
Object	0,678198185	2,515392412	3,701890226
Param	0,682173928	2,714439470	3,882733933
Min	0,648176540	2,059638782	3,457200403
Q1	0,676022479	2,0888628	3,563528212
Median	0,678556269	2,155977107	3,658611690
Q3	0,691908103	2,490195232	3,748590334
Max	0,773343804	2,714439470	3,965579806
Box 1 (hidden)	0,676022479	2,0888628	3,563528212
Box 2 (lower)	0,002533790	0,067114307	0,095083478
Box 3 (upper)	0,013351834	0,334218125	0,089978644
Whisker Top	0,081435702	0,224244238	0,216989472
Whisker Bottom	0,027845939	0,029224018	0,106327809
Range	0,125167264	0,654800688	0,508379403
Factor of Range Differance between Phantom and Firefox			4,061600344
[25-75] percentile range	0,015885624	0,401332432	0,185062122